CANADIAN THESES ON MICROFICHE

I.S.B.N.

THESES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

NL-339 (r. 82/08)

Canada

40.

0-315-15950-2

National Library of Canada | Bibliothèque nationale du Canada

Canadian Theses Division | Division des thèses canadiennes

Ottawa, Canada
K1A 0N4

63859

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

Francis Jeffry Pelletier

Date of Birth — Date de naissance	Country of Birth — Lieu de naissance
Jan 30 1944	USA

Permanent Address — Résidence fixe

10516-17 Ave
Edmonton, Alberta T6J 5B5

Title of Thesis — Titre de la thèse

Completely Non-Clausal, Completely Heuristically Driven, Automatic Theorem Proving

University — Université

Univ Alberta

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

M Sc

Year this degree conferred — Année d'obtention de ce grade	Name of Supervisor — Nom du directeur de thèse
1983	Lenhart K. Schubert

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

Date	Signature
25 April 1983	Francis Jeffry Pelletier

NL-91 (4/77)

THE UNIVERSITY OF ALBERTA

COMPLETELY NON-CLAUSAL, COMPLETELY HEURISTICALLY DRIVEN, AUTOMATIC THEOREM PROVING

by

Francis Jeffry Pelletier

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

Department of Computing Science

EDMONTON, ALBERTA

Spring 1983

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR Francis Jeffry Pelletier

TITLE OF THESIS COMPLETELY NON-CLAUSAL, COMPLETELY HEURISTICALLY DRIVEN, AUTOMATIC THEOREM PROVING

DEGREE FOR WHICH THESIS WAS PRESENTED MASTER OF SCIENCE

YEAR THIS DEGREE GRANTED Spring 1983

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(SIGNED) Francis Jeffry Pelletier

PERMANENT ADDRESS:

10516 - 17 Avenue

Edmonton, Alberta

T6J 5B5

DATED April 25 19 83

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled COMPLETELY NON-CLAUSAL, COMPLETELY HEURISTICALLY DRIVEN, AUTOMATIC THEOREM PROVING submitted by Francis Jeffry Pelletier in partial fulfilment of the requirements for the degree of MASTER OF SCIENCE.

L.K. Schubert

Supervisor

Date.....Apr. 20/83.....

ABSTRACT

Automatic theorem proving, that is, theorem proving by computer, has a wide variety of applications. Chapter I takes a light-hearted look at some of these ways in order to motivate the search for a good theorem proving system. Chapter II is a theoretical discussion of various issues in logic such as the distinction between "semantic" and "natural deduction" systems of logic, and the importance of such notions as completeness and the deduction theorem. This chapter also describes the resolution rule of inference and a natural deduction system due to Kalish & Montague (1964). Chapter III is a brief survey of the history of automatic theorem proving from the Logic Theorist of 1957 to recent connection graph resolution-based systems and certain current natural deduction systems. Chapter IV is a catalog of problems that attend these systems. Chapters V and VI describe a theorem proving program based on the Kalish & Montague system. This system attacks the proof of a theorem by employing heuristics or strategies which are based on the type of formula to be proved and on previous lines of the proof, and unlike resolution systems, does not require any pre-processing of formulae to put them into clause form. Chapter VII describes the theorems which can be proved and compares them to the output of previous theorem provers. There are some theorems which cannot be stated in the language treated by the present theorem prover which are statable in other theorem proving systems. But there are no

theorems which are statable in both systems that are provable by any of the other systems which the present system cannot also prove; however, the reverse is not true for any of the published theorem provers. The final chapter describes some future extensions to the system, which work is currently underway. Two Appendices give: (a) printouts of proofs of a selection of theorems, with a commentary on how this compares to other systems, and (b) a pidgin ALGOL statement of the heuristics employed.

Acknowledgements

Dan Wilson, currently a Faculty Service Officer in the Department of Computing Science at the University of Alberta, deserves a major share of the credit for the program called "THINKER" in this thesis. Together we wrote the program and improved on it. Generally speaking, Dan was responsible for the data structures and low-level routines while I was responsible for the actual proof strategy. But this is only "generally speaking" -- we have, on numerous occasions, discussed all aspects of THINKER. An informal paper on THINKER, prepared for an interdisciplinary audience, was presented in January 1982 at the University of the South Pacific in Suva, Fiji, to the Conference on Thinking, and is to be published as Pelletier & Wilson (1983).

It was as a course project for Prof. Jeffrey Sampson of the Computing Science Department at the University of Alberta that Dan and I originally wrote a prototype of THINKER. I wish to thank him for his encouragement of that project and for his careful reading of and comments on the present work. Prof. W. David Sharp of the Philosophy Department of the University of Alberta supplied me with lists of "tricky" theorems for THINKER to prove during the development stage. I also appreciate his diligent reading of and corrections to the present work. Finally, but not least, is the debt of thanks I owe my supervisor, Prof. Lenhart K. Schubert of the Computing Science Department of the

University of Alberta, who first interested me in computer science in general and computational linguistics in particular. He has discussed the present system and its underlying assumptions and justification on numerous occasions, making a large number of suggestions. I have not always followed these suggestions, so any shortcomings in this work should not be traced to him. He also invested some of his computer time (under NSERC grant A8818) to further development of this system, for which thanks are hereby given.

Finally, I wish to thank the University of Alberta for its enlightened policy of encouraging faculty members to take courses in other departments (and work toward degrees without meeting the usual full-time residency requirements) by giving a tuition grant for this purpose. The Department of Computing Science is also hereby given thanks for taking in a wayward philosopher.

Table of Contents

Chapter — Page

I. WHY BOTHER? 1

II. LOGIC: SOME THEORETICAL REMARKS 12

A. Introduction: Formulae, Definitions, and Abbreviations 12

B. A Classification of Systems of Logic 17

C. The "Axiomatic vs. Semantic vs. Natural Deduction" Division 28

D. Arguments and Theorems 37

E. The Deduction Theorem 40

III. A BRIEF SURVEY OF AUTOMATIC THEOREM PROVING: METHODS AND STRATEGIES 43

A. Soundness, Completeness, and Decidability 43

B. Early Heuristic Approaches 48

C. The Decline of Heuristics 49

D. The Re-emergence of Heuristics 52

E. The Retreat from Resolution, I 67

F. The Retreat from Resolution, II 71

IV. METHODOLOGICAL AND PRACTICAL PROBLEMS IN AUTOMATIC THEOREM PROVING 77

A. Introduction 77

B. Some Theoretical Remarks about Theorem Proving by Machine 82

C. Some Problems with the Resolution Strategies ..87

D. Resolution in General 92

E. Problem Reduction Format 95

F. Getting into Clause Form 100

G. Bledsoe's "natural" systems: A critique 102

H. Natural Deduction and the Problem Reduction Format 104

V. THE GUTS: DATA STRUCTURES AND LOW LEVEL ROUTINES 109

A. Introduction 109

B. Explicit Manipulation of Formulae 110

C. The Proof Matrix 113

D. Antecedent Lines and Templates 114

E. Goals 116

F. SIMPLEs and FINDing 117

G. TESTing and SEARCHing 119

VI. THE BRAINS: HEURISTICS 121

VII. SOME COMPARISONS AND DISCUSSION 133

A. The Logic Theorist and the British Museum Algorithm 133

B. Bledsoe's "Natural" Systems 135

C. The Full Propositional Logic 137

D. The Predicate Logic 138

E. The UI/EI Problem, I 141

F. The UI/EI Problem, II 146

G. The UI/EI Problem, III 149

H. The UI/EI Problem, IV 152

I. The Logic of Set Theory 155

J. Andrew's Challenge 156

K. Schubert's Steamroller 158

L. Can THINKER Prove the Steamroller? 160

VIII. SOME DIRECTIONS FOR THE FUTURE 175

A. Improvements to the Proof Strategy 175

B. Identity and Functions 178

C. Natural Language Processing and Other Areas ..181

IX. APPENDIX I: THINKER's PROOFS OF SOME THEOREMS ...183

X. APPENDIX II: A FLOW CHART OF THINKER's PROOF STRATEGY 266

A. ONESTEP and SIMPLEPROOF 266

B. Initialization 268

C. PROOF 269

XI. REFERENCES 278

I. WHY BOTHER?

Polly Programmer has just written a 5000 line program in ANSI 77 FORTRAN. No matter how talented Polly is, she will not be able to foresee all the ins and outs of her program. One way to try to find out -- the way she was taught at university -- would be to run her program on a wide variety of test data to guarantee that it performs as desired in these cases. But the years since university have taught Polly that she is remarkably bad at guessing what sorts of input those silly users will want to enter and how her program will behave. What Polly really wants is a *program verifier*: a program which will mechanically prove that her program satisfies some specification, or produces the same output as another program, or can be executed within certain time and space bounds. To do this, there must be a formal program semantics given for ANSI 77 FORTRAN, a formal logical theory, and an automatic theorem prover for that theory. Such a verifier reduces the question of whether the program has such-and-such a property to the question of whether so-and-so formulae are theorems of the system.

Jim Faculty, Jr. wants tenure in the mathematics department at the University. Jim wishes to be the "idea man" for an interactive mathematics prover. He would like to be able to type in PROVE(FERMAT) and have his automatic

theorem prover work on the problem for a while. If it should get "stuck", he would like to be able to suggest guidance in the form of new premises to be added or lemmas to be proved first. He reasons that, far from subverting the creative aspect of mathematics, much less the tenure system, this man-machine interaction will free the ordinary working mathematician from all the low-level drudgery of getting unimportant details correct, and allow him to concentrate on the truly creative aspects of his field. In this he is following the famous solution to the four-colour problem.

MAO, the Mechanical Admissions Officer at the University, is reading over letters of recommendation from undergraduate teachers concerning prospective graduate students. One such letter reads

> Dear Sir, Candidate X's attendance at my classes has been excellent and he has been known to produce artifacts which bear a certain semblance to good computer programs. Sincerely, Prof. Y

MAO's *conversational implicature* subprogram reasons as follows: "Nothing in the literal meaning of Prof. Y's letter is relevant to X's getting into graduate school. However, if Prof. Y really had not wanted to say anything he would not have written. And he must be able to say something relevant since X was his pupil. Furthermore, he knows that more information is wanted, since he has done this before. He must therefore be wishing to impart information that he is reluctant to write down. There is a briefer and clearer, but

nearly synonymous, way to express "produce artifacts which ....", namely "programs well." The most obvious supposition is that Y is drawing attention to some striking difference between X's performance and that to which "programs well" usually applies. It is most likely that X's performance has some hideous defect. I therefore conclude that, in Y's opinion, X is not a good candidate for graduate study."

Charlie Cheater, a student at the University, has even more grand desires than Polly Programmer. He wants to be able to specify an algorithm and have an *automatic program generator* produce source code in PASCAL. Charlie does not view this as cheating. "After all," he asks, "isn't this just what compilers do? Weren't the original FORTRAN and COBOL compilers called 'automatic program generators'? What's the difference between generating machine code from a high level language and generating high level code from a super-high level language?" With this analogy in mind, Charlie has written a "compiler" which takes the language of his algorithm and (perhaps after repeated passes) converts his sentences into constructs of PASCAL. In his "compiler", the expressions which occur during this transformation process not only specify the target program, but also specify conditions to be proved as well as conditions to be made true. For an expression which specifies a (sub)program, the goal is to convert that (sub)program into PASCAL; for an expression which is a condition to be proved, the goal is to convert it into the logical constant **true**; and for an

expression which is a condition to be made true, the goal is to construct a program that will make the condition true. In achieving these goals, including the goal of writing PASCAL, Charlie's "compiler" creates a tree of goals and subgoals. It then proceeds to establish these goals by invoking a theorem prover which guarantees that each goal has been established.

Charlie's friend Larry Lazy wants to be able to specify only input-output relations and have the automatic programmer produce a PASCAL program that will also exhibit those input-output relations. As clever as Charlie was to write his "compiler", Larry is even cleverer. He notices that what he needs to do is prove this as a theorem:

$$(Ax)[Px \rightarrow (Ey)Qxy] \qquad (1)$$

where x ranges over input variables and P is the predicate that the input is expected to satisfy, y ranges over output variables and Q specifies the relation between each input variable and its associated output variable after the execution of the desired program. Now, given sufficient other axioms so that (1) is provable, the construction of the proof of (1) will allow the extraction of a program. For example, certain constructs in the proof will produce conditional statements, others will produce sequential statements, and uses of induction axioms will produce recursive loops.

Felicity Findout is writing a large data base retrieval question answering system. In her earlier investigations she

discovered that the typical questions to be answered fell into three categories: (a) those questions requiring yes/no answers, (b) those questions requiring the name of some entity (state,...) such as WH-questions, and (c) those questions requiring the specification of a sequence of actions such as HOW-questions. Early on Felicity decided that the natural way to deal with type (a) questions was to represent the data base as a set of predicate logic statements, the question as a declarative statement, and invoke a theorem prover to try first to deduce the question-statement from the data base. If successful, answer "yes", if not then try to deduce its negation from the data base. If successful, answer "no"; otherwise the answer is "don't know". Felicity uses a *variable tracing* method to handle type (b) questions. Given, for instance, that Dr. Smarts is the Chairman of the Computer Science Department, this is represented in the data base as

$$C(s,csd) \qquad (2)$$

And to answer the question "Who is Chairman of the Computer Science Department?", Felicity's program tries to prove that

$$(Ex)C(x,csd) \qquad (3)$$

follows from the information in the data base. It does of course: assume that (3) is false and a contradiction will ensue from (2) by instantiating the (universally quantified) assumption that (3) is false to s. All that is needed then is a way of finding out that it was the instance s which resulted in the contradiction -- some "variable tracing"

mechanism. Felicity has yet to fully implement a solution to questions of type (c), but her solution to the "monkey-banana" problem (which builds upon her type (b) "variable tracing" mechanism) gives her cause for optimism.

> A monkey wants to eat a banana that is suspended from the ceiling of a room. The monkey is too short to reach the banana; however, it can walk around the room carrying a chair that is in the room, and it can climb the chair to reach the banana.

The relevant predicates and functions are:

> P(x,y,z,w): in state w, the monkey is at x, the banana at y, and the chair at z
>
> R(x): in state x, the monkey can reach the banana
>
> f(x,y,z): the state attained if the monkey is initially in state z and walks from x to y
>
> g(x,y,z): the state attained if the monkey is initially in state z and walks from x to y carrying the chair
>
> h(x): the state attained if the monkey is initially in state x and climbs the chair

Assume the initial position of the monkey is at *a*, the banana at *b*, the chair at *c*, and the monkey is in state *s*. Then the data base contains

$$(Ax)(Ay)(Az)(Aw)[P(x,y,z,w) \rightarrow P(z,y,z,f(x,z,w))]$$

$$(Ax)(Ay)(Az)[P(x,y,x,z) \rightarrow P(y,y,y,g(x,y,z))]$$

$$(Ax)[P(b,b,b,x) \rightarrow R(h(x))]$$

$$P(a,b,c,s)$$

That is, respectively: in any state the monkey can walk from

x to *z*; in a state where the monkey is beside the chair located at *x*, it can carry the chair to any place *y*; if the chair and the monkey are both under the banana, then it can climb the chair and get the banana; and finally a statement of initial conditions. Felicity's program is to prove (Ex)Rx, and "trace" the appropriate instance. Her program proves it and yields

h(g(c,b,f(a,c,s)))

as the relevant instance, i.e., the state resulting from climbing the chair after having walked from *c* to *b* carrying the chair after getting into the state resulting from walking from *a* to *c* out of the initial state *s*.

Robbie Robot is ordered to move a newly delivered computer terminal *t* from the loading dock *l* to a point in the terminal room *r*, namely the back corner, *b*(*r*). In Robbie's knowledge base is a representation of the physical relationships between *l* and *r*, to wit, that they are each five meters square, that they are located some 50 meters apart, that they are joined by a corridor *c*, that the point where *c* and *l* join contains a closed door d_1, that the point where *c* and *r* join contains a closed door d_2, that any route between *l* and *b*(*r*) except the corridor contains "obstructions" (viz., walls) and it has no action with which it can change this obstructiveness condition, and finally that d_1 and d_2 are the only "obstructions" along the corridor route between *l* and *b*(*r*). Robbie also has in its knowledge base a repertoire of primitive actions which

include moving one meter in any direction so long as there is no obstruction in that new location, and opening doors if they are located within one meter and it is not holding anything. For ease of computation, Robbie is also endowed with a higher order action of the form "if there is no obstruction between Robbie's present location and location *x*, then Robbie can move to *x*". Robbie also has in its possession a variety of "general knowledge": That if two physical objects are located further apart than their size, then they are not identical; that an open door is no obstruction; that if Robbie is carrying *x* and Robbie moves from *y* to *z*, then *x* moves from *y* to *z*; that if *x* is in *y* and $y \neq z$ then *x* is not in *z*; and if there is no obstruction between *x* and *y* and there is no obstruction between *y* and *z*, then there is no obstruction between *x* and *z*.

How is Robbie to get *t* from *l* to *b*(*r*))? Well, what Robbie needs to do is become convinced that *t* is in *b*(*r*). Being efficient, Robbie will first try to become convinced that *t* is already at *b*(*r*). So Robbie says to itself: *l* and *r* are five meters long and located 50 meters apart, 50>5, hence $l \neq r$. *b*(*r*) is in *r*, hence $l \neq b(r)$. But *t* is in *l* hence *t* is not in *b*(*r*). Having convinced itself that there's work to be done, Robbie goes through its repertoire of actions and discovers that if Robbie is at *b*(*r*) and carrying *t* then *t* is in *b*(*r*). Robbie fails at its proof that it is at *b*(*r*) and carrying *t*, and so again goes through its repertoire of actions to discover that if there were no obstructions

between *l* and *b(r)* it could get to *b(r)* carrying *t*. So Robbie next tries to convince itself that there are no obstructions. This fails of course, but in discovering this failure, Robbie also discovers that if it were in the corridor within one meter of the door and not holding anything, it could open d_2 and there would be no obstruction between *c* and *b(r)*. A similar consideration makes Robbie discover that if it were in *l* and within one meter of the door and not holding anything, it could open d_1 and there would be no obstruction between *l* and *c*. From these Robbie concludes that these two actions would make there be no obstruction between *l* and *b(r)*. Given that, Robbie finally concludes that in such a situation it can pick up *t* (if it is within one meter) and carry it to *b(r)*, and then *t* will be in *b(r)*. Thereby Robbie makes true the statement: *t* is at *b(r)*. Robbie then carries out the following sequence of actions: (a) Go from start position to d_1, (b) Open d_1, (c) Go from d_1 to d_2, (d) Open d_2, (e) Go from d_2 to start position, (f) pick up *t*, (g) go from start position to *b(r)*. (With a somewhat different control structure Robbie might perform this perhaps less efficient sequence: (a') Pick up *t*, (b') Go from start position to d_1, (c') set *t* down, (d') Open d_1, (e') Pick up *t*, (f') Go from d_1 to d_2, (g') set *t* down, (h') Open d_2, (i') Pick up *t*, (j') Go from d_2 to *b(r)*.)

Sammy Psycho is investigating human problem solving techniques and wants to simulate the mechanisms by which

people go about their everyday business, in order to better understand these mechanisms. Sammy wants to find out what it is about the human mind that sets it apart from other animals. Throughout history it has been taken to be the hallmark of being human that humans are rational and think -- and this was always expanded into: is able to employ syllogistic reasoning, is able to solve mathematical problems, and (sometimes) is able to use language. In the last 20 years it has been common to mark off human performance from computer performance in terms of "heuristics" vs. "algorithms". A computer, so it is claimed, blindly follows a (human-devised) algorithm and therefore cannot be counted rational, no matter what its performance. Humans, on the other hand, have a variety of heuristics or strategies at their disposal and engage in goal-directed thought by employing these strategies so long as they appear to be leading toward the goal, and switching to another strategy when the previous one appears not to be succeeding. Sammy is not convinced that there is really a distinction between "blind algorithms" and "heuristic strategies", he thinks that they merge into one another and that the real difference between them is a subjective impression of unsureness of success in the case of heuristics. But for now he is going to accept the received distinction and try to display a theorem proving program which uses exclusively what everyone would recognize as heuristic strategies. This having been done, Sammy thinks he will be in a position to

simulate the phenomenon of truly human rationality.

Sources: Polly Programmer's desires were fanned recently by a close reading of Manna & Waldinger (1977) and Boyer & Moore (1979), as were Charlie Cheater's. Jim Faculty, Jr. has been reading Bledsoe & Bruell (1974), Guard *et al* (1969), and the discussion of the proof of the four-colour problem by Appel & Hakin (1977). MAO's example letter is close to some discussed by Grice (1975); some steps toward implementation of this sort of example are taken by Lehnert (1977) and Reiter (1978). Larry Lazy spent some considerable time reading Biermann (1976), Elschlager & Phillips (1979), Goad (1980), Bibel (1979), and Guiho & Greese (1980). Felicity Findout's program has been guided in its development by, first, the work of Green (1969) and more particularly the examples and programs found in Chapter 11 of Chang & Lee (1973). Robbie Robot's program is a development of STRIPS (itself a development of GPS, see Newell & Simon (1963) and Ernst & Newell (1969)), for which see Fikes & Nilsson (1971). The line of development goes through PLANNER-like languages (see Bobrow & Raphael (1974) and Derksen, Rulifson & Waldinger (1972)), and more precisely to ABSTRIPS-style systems for hierarchical planning (Sacerdoti 1974). Sammy Psycho is obviously a student of Newell & Simon (1972). All of them have read Nilsson (1980).

II. LOGIC: SOME THEORETICAL REMARKS

A. Introduction: Formulae, Definitions, and Abbreviations

A logic is a system of symbols organized in some specified way. Of course, there are a variety of ways to organize these symbols; I will use the following definition of *formula* as common to all the systems considered. The differences amongst the various systems will therefore come out in their respective definitions of *proof*. Still later we shall consider various augmentations of the notion of formula and concomitant extensions of the definition of proof.

A *variable* is one of x_i, y_i, z_i, w_i, where i is a positive integer

A *constant* symbol is one of a_i, b_i, c_i, d_i, where i is a positive integer

A *function symbol* is one of f_i^j, g_i^j, h_i^j, where i and j are positive integers (f_i^j is called a j-place function symbol)

A *term* is either a variable or a constant or a j-place function symbol followed by an opening parenthesis, followed by j terms separated by commas, and followed by a closing parenthesis[1]

A *predicate* is one of P_i^j, Q_i^j, R_i^j, S_i^j, where j and i are non-negative integers. P_i^j is called a j-place

---

[1]Constants can be assimilated to function symbols by allowing j (in the function symbol definition) to be non-negative, and then constants are 0-place function symbols.

predicate).[2]

A *quantifier* is one of A and E.

Formulae:

(1) If ¶ is a j-place predicate, then $¶(\alpha_i,\ldots,\alpha_j)$ is a formula.

(2) If ø and ψ are formulae, so are (ø→ψ), (ø&ψ), (ø+ψ), and (ø↔ψ)

(3) If ø is a formula, α is a variable, and *Q* is a quantifier, then ¬ø and (*Q*α)ø are formulae.

Some other ancillary notions and abbreviatory devices which shall be used below are: propositional formula, atomic formula, the main connective of a formula, conditional, conjunction, disjunction, biconditional, negation, quantified formula, subformula, free/bound occurrence of a variable, literal, conjunctive normal form, disjunctive normal form, prenex normal form, skolem normal form, clause form, and clause.

A *propositional formula* is a formula wherein (a) there are no quantifiers and (b) every predicate occurring in it is 0-place. An *atomic formula* is of the form: a j-place predicate followed by j terms. As an abbreviatory device, I shall use lower case letters p, q, r, s without sub- and superscripts to describe atomic propositional formulae. If a formula is not atomic then it is *complex* and hence is described by one of the rules (2) or (3) given above for formulae. If it is by rule (2a), then the *main connective* is

---

[2]A proposition is a 0-place predicate.

→ and it is a *conditional*; if it is by rule (2b), then the *main connective* is & and it is a *conjunction*; if it is by rule (2c), then the *main connective* is + and it is a *disjunction*; if it is by rule (2d), then the *main connective* is ↔ and it is a *biconditional*; if it is by rule (3a), then the *main connective* is ¬ and it is a *negation*; if it is by rule (3b), then the *main connective* is the quantifier and it is a *quantified formula*. These latter are broken down into *existential formulae* where the quantifier symbol is E and *universal formulae* where the quantifier symbol is A. If ø is a quantified formula, then it starts with a *quantifier phrase* which is an open parenthesis followed by a quantifier symbol followed by a variable followed by a closing parenthesis. The variable in the quantifier phrase is called the *variable of quantification*. The *scope* of the quantifier phrase is the (unique) formula following the quantifier phrase. A *subformula* of ø is a formula which is a physical part of ø. (Note that ø is a subformula of itself). An occurrence of (variable) α in formula ø is *bound* if and only if either (a) it is in a quantifier phrase or (b) it occurs in a subformula of ø which is a quantified formula and the variable of quantification is α. It is *bound by* that quantifier phrase. An occurrence of a variable is *free* if and only if it is not bound. As another abbreviatory device, I shall eliminate subscripts on variables, predicates and constants except when they are needed for distinguishing one from another. Function symbols and predicates will have

their superscripts deleted when it is clear how many arguments they take. A *literal* is either an atomic formula or the negation of an atomic formula.

Disjunctive and conjunctive formulae are taken to obey the following associative laws:

$((\phi_1 \& \phi_2) \& \phi_3) = (\phi_1 \& (\phi_2 \& \phi_3))$

$((\phi_1 + \phi_2) + \phi_3) = (\phi_1 + (\phi_2 + \phi_3))$

As an abbreviatory device then, one can omit the internal parentheses and widen the definition of conjunctions and disjunctions to include

$(\phi_1 \& \phi_2 \& \phi_3)$

$(\phi_1 + \phi_2 + \phi_3)$

and further to include as many *conjuncts* or *disjuncts* as we wish. A formula is in *conjunctive normal form* (*disjunctive normal form*) if and only if it is a conjunction (disjunction) where all the conjuncts (disjuncts) are disjuncts (conjuncts) of literals. (Note that as a special case a single literal is both in conjunctive normal form and disjunctive normal form). It is a well known result that every quantifier-free formula can be put into an equivalent conjunctive normal form and an equivalent disjunctive normal form.[3] By "equivalent" here is meant that the following formula will be a theorem of such a standard logic (see below for the notion of a theorem).

$\phi \leftrightarrow \phi'$

where ϕ' is the conjunctive normal form or disjunctive

---

[3]Under standard interpretations of the logic, which is all that we shall be concerned with here.

normal form of ø. A formula is in *prenex normal form* if it is of the form

$$(Q_1\alpha_1)(Q_2\alpha_2)\ldots(Q_m\alpha_m)\text{ø}$$

where the Q's are quantifiers, the α's are all distinct, and ø contains no quantifiers. That is, all quantifiers are "in front of the matrix." It is again well known that every formula has an equivalent prenex normal form. Given a formula in prenex normal form, one can arrive at a *skolem normal form* by following the following procedure. Start with the innermost quantifier phrase, working outwards one by one to the outermost quantifier phrase, and do the following: (a) if the quantifier phrase is universal, delete it and move to the next outermost quantifier phrase, (b) if the quantifier phrase is existential and there are no universal quantifier phrases remaining to process, replace each occurrence of the variable bound by that quantifier phrase by a constant that has never appeared in the formula up to this stage, delete the quantifier phrase, and move on to the next outermost quantifier phrase, (c) if the quantifier phrase is existential and there are n more universal quantifier phrases left to process, replace each occurrence of the variable bound by that existential quantifier phrase by a new n-place function symbol and make the n arguments of that function symbol be the n variables mentioned in the unprocessed universal quantifier phrases, delete the existential quantifier phrase, and move on to the next outermost quantifier phrase. It can be shown that the

resulting formula is a theorem if and only if the starting prenex normal form formula was a theorem. (But note that they are not equivalent in the sense defined above). Thus every formula ø has a skolem normal form which is a theorem if and only if ø is a theorem.

If the matrix of the prenex normal form was in conjunctive normal form, then the resulting skolem normal form will be also. (Equivalently, the resulting skolem normal form has an equivalent conjunctive normal form). Such a formula is said to be in *clause form*, and each of the conjuncts is a *clause*. Thus each clause is a disjunction of literals.

B. A Classification of Systems of Logic

In the abstract, it is traditional to divide systems of logic into three sorts: axiomatic, "semantic", and natural deduction. The reason for this division seems to be based on a variety of considerations, which do not always cohere in such a way as to give univocal results. In the next section I will briefly consider the reasons one might offer in support of this classification, but first we should look at some examples so as to have some background exemplars to discuss.

An axiomatic system of logic takes certain formulae as "given" (in the sense of requiring no other justification), gives a set of "rules of inference" (methods of transforming one (or more) formulae into another), and defines a *proof* as

an ordered (finite) set of formulae, each one of which is an axiom or follows from previous (in the ordering) formulae by a rule of inference. To give an example of a proof in a typical propositional axiomatic system, consider the system P1 of Church (1956). Included in the axioms are

A1: (p→(q→p))

A2: ((s→(p→q))→((s→p)→(s→q)))

and the two rules of inference,

MP: from (A→B) and A, infer B

Sub: from A, if *b* is a propositional variable in A, infer the result of replacing all occurrences of *b* in A by a formula B

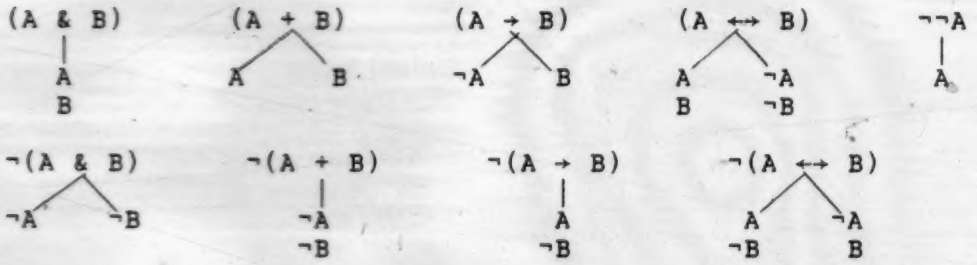
A proof of the theorem (p→p) in this system would be

1.	((s→(p→q))→((s→p)→(s→q)))	A2
2.	((s→(r→q))→((s→r)→(s→q)))	1,Sub (r for p)
3.	((s→(r→p))→((s→r)→(s→p)))	2,Sub (p for q)
4.	((p→(r→p))→((p→r)→(p→p)))	3,Sub (p for s)
5.	((p→(q→p))→((p→q)→(p→p)))	4,Sub (q for r)
6.	(p→(q→p))	A1
7.	((p→q)→(p→p))	5,6MP
8.	((p→(q→p))→(p→p))	7,Sub ((q→p) for q)
9.	(p→p)	6,8MP

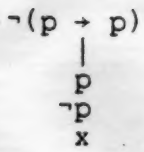
The extension of the axiomatic method to the predicate calculus is accomplished by adding further axioms or rules of inference.

"Semantic" systems of logic are so-called because they attempt to mirror the intended semantical interpretation in the system of logic itself. For the propositional logic, this intended semantical interpretation is just the truth table; and consequently, to prove whether a formula A is a theorem or not, it is customary in these systems to

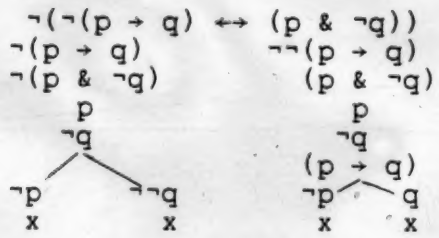
introduce devices which enable one to find out whether the assumption that A is *not* a theorem would also require that some atomic sentence and its negation both be assigned **true**. This is normally done by "breaking down" the formula ¬A into simpler and simpler components. Many of these methods can easily be represented by trees. Jeffrey (1967) has the following system of rules for tree construction:



where the intuitive idea behind a rule is that we are interested in "ways the complex formula might be true." The definition of a proof of conclusion C is: (1) ¬C is the root node, (2) if one wishes to have premises, they are added to the root node, (3) if a rule of inference is applied to a formula B which occupies a node of the tree, the result of the rule of inference is represented in every "uncancelled" branch that B dominates, (4) any branch which contains an atomic formula and also its negation is "cancelled" by putting an "x" at the bottom of the branch, (5) every branch is cancelled. C is a theorem if and only if there is a proof of it. The proof of the formula (p→p) is very simple in this system. We put ¬(p→p) as the root node and use the rule for ¬(A→B):



A somewhat more interesting theorem is DeMorgan's $(\neg(p \to q) \leftrightarrow (p \& \neg q))$ which is proved:



It is quite clear here (as opposed to the axiomatic system) what the strategy is: one assumes the (alleged) theorem to be false and breaks it down into simpler and simpler components by the truth-preserving rules (the branched formula is true if and only if at least one of its subbranches is true). Since the resulting formulae get shorter and shorter, the method (in the propositional calculus) is guaranteed to halt.

In computerized theorem proving, the most commonly used method is "resolution" -- a variant of the semantic methods. Here (in the propositional logic case) one negates the formula to be proved and represents it by its equivalent clause form. (Recall that this is obtained if the formula is converted to skolem normal form and then to a conjunction of disjunctions of literals; each conjunct being called a clause). Each clause (which is itself in disjunctive normal form) is written on a separate line and the one rule of

inference, "resolution",[4] is used. If premises are to be used, they too are converted to clause form and each clause is written on a separate line. The rule of resolution is (in its simplest statement)

$$A_1 + B_1 + \ldots + P_1 + \ldots + Z_1$$

$$A_2 + B_2 + \ldots + \neg P_1 + \ldots + Z_2$$

$$A_1 + B_1 + \ldots + Z_1 + A_2 + B_2 + \ldots + Z_2$$

where: each of the lines is in clause form and the conclusion (a new clause) has no mention of P_1 or its negation (it has been "resolved out"). This new clause which has been created is added to the list of clauses and the resolving procedure is continued. If the original formula was a theorem, then eventually the method will yield a *null resolvent* -- the "empty formula", a formula with no subformulae. (The rule is usually generalized to apply to an arbitrary number of premises at one swoop.) Clearly the resolution method is semantic in nature: one is trying to discover whether the purported statement is necessarily true by looking at ways its negation might be true. If none are found (null resolvent), the negation can't be true and so the original statement must be.

These semantic methods can be extended to the (non-decidable) predicate calculus in various ways. Jeffrey (1967) adds branching rules for quantifiers, but these rules are not effective in the sense that they needn't ever be

---

[4] Introduced into the literature by Robinson (1965).

used again.[5] Another way would be to convert the formula into a skolem normal form. This resulting (non-quantified) formula can now be treated in various ways. One could apply the tree method of above,[6] or one could continue to use the resolution procedure by introducing a special understanding of what variables can resolve against which and generate the null resolvent. Such methods will be considered in the next chapter. Even in the complex case of quantifiers it should be clear that the strategy is *semantic*: quantifiers are interpreted -- existential quantifiers not in the scope of a universal are replaced by a name (the thing in the model that the sentence asserts the existence of), existential quantifiers in the scope of a universal quantifier are replaced by a function of the things named in the model by the universal quantifiers, and so on. Finally we merely look at the possible co-truth of the atomic formulae.

A natural deduction system is like the semantic systems and unlike the axiomatic systems in that it has no unjustified statements (axioms) and in that it has a large number of rules of inference; however, it is unlike the semantic systems in that it does not attempt to "break formulae down" into simple components and evaluate their possible co-truth. Rather, the rules of inference are supposed to correspond to psychologically plausible modes of

---

[5] All formulae in a proof have the property that they need be "branched" at most once, except for universally quantified formulae.

[6] This would require that each clause be considered a universally quantified formula over the variables mentioned in the matrix.

reasoning. A proof is a method of breaking down a formula into "what you can assume" and "what still needs to be proved", together with methods to actually do some of the "proving". There are a number of these natural deduction systems in the literature; I shall here present (and later employ) the one found in Kalish & Montague (1964). For the propositional logic, the *rules of inference* are:

$$\frac{A}{A}\ (R); \qquad \frac{A}{\neg\neg A} \text{ and } \frac{\neg\neg A}{A}\ (DN); \qquad \frac{(A\&B)}{A} \text{ and } \frac{(A\&B)}{B}\ (S);$$

$$\frac{(A\rightarrow B)}{\frac{A}{B}}\ (MP); \qquad \frac{(A\rightarrow B)}{\frac{\neg B}{\neg A}}\ (MT); \qquad \frac{A}{\frac{B}{(A\&B)}}\ (Adj); \qquad \frac{(AvB)}{\neg A}{B} \text{ and } \frac{(AvB)}{\neg B}{A}\ (MTP);$$

$$\frac{(A\rightarrow B)}{(B\rightarrow A)}{(B\leftrightarrow A)}\ (CB); \qquad \frac{(A\leftrightarrow B)}{(A\rightarrow B)} \text{ and } \frac{(A\leftrightarrow B)}{(B\rightarrow A)}\ (BC); \qquad \frac{A}{(AvB)} \text{ and } \frac{A}{(BvA)}\ (Add)$$

which abbreviations stand for, respectively R: Repetition, DN: Double Negation, S: Simplification, MP: Modus Ponens, MT: Modus Tollens, Adj: Adjunction, MTP: Modus Tollendo Ponens, CB: Conditionals to Biconditional, BC: Biconditional to Conditional, Add: Addition. These are all taken to be psychologically plausible modes of reasoning. An *antecedent line* is defined as a line which is earlier in the proof and neither boxed nor containing an uncancelled 'show' (both defined below). A *proof* is defined as:

1. If ø is a formula, then

   Show ø

   can occur as a line. (The 'show' is *uncancelled*. Intuitively we are setting the task of proving ø).

2a. If Show ø occurs as a line then ¬ø can occur as the next

line ("assume the negation").

2b. If Show ¬ø occurs as a line, then ø can occur as the next line.

2c. If Show (ø→ψ) occurs as a line, then ø can occur as the next line ("assume the antecedent").

3. If ø follows from antecedent lines by a rule of inference, then ø may be entered as the next line.

4. If the proof has a subpart which looks like

```
Show ø
   X1
   .
   .
   .
   Xn
```

and (a) there are no uncancelled 'Show' among X1...Xn, and (b) either ø occurs unboxed (defined below) among X1...Xn, or else both ψ and ¬ψ occur unboxed among X1...Xn, then

```
*Show ø
| X1
| .
| .
| .
| Xn
```

can be the next step in the proof (X1...Xn are now *boxed* -- and thus are no longer antecedent, and the 'Show' line is *cancelled* and now antecedent (intuitively, the lines in the box constitute a proof of ø).

5. If the proof has a subpart which looks like

```
Show (ø→ψ)
   X1
   .
```

.
.
Xn

and (a) there are no uncancelled "Show" among X1...Xn, and (b) ψ occurs unboxed among X1...Xn, then

*Show ($\phi \rightarrow \psi$)
| X1
| .
| .
| .
| Xn

may occur as the next step of the proof.

6. A premise may be entered anywhere in the proof.

7. The formula ϕ is proved if it occurs unboxed in a proof and there are no uncancelled "Show" in the proof.

This natural deduction system is extended to the predicate calculus by adding rules for Quantifier Negation (QN), Existential Instantiation (EI), Universal Instantiation (UI), Existential Generalization (EG), and another method of boxing and cancelling called "universal derivation". These rules of inference are:

$$\frac{\neg(A\alpha)\phi}{(E\alpha)\neg\phi} \text{ and } \frac{\neg(E\alpha)\phi}{(A\alpha)\neg\phi} \text{ and } \frac{(A\alpha)\neg\phi}{\neg(E\alpha)\phi} \text{ and } \frac{(E\alpha)\neg\phi}{\neg(A\alpha)\phi} \quad \text{(QN)}$$

$$\frac{(E\alpha)\phi}{\phi'}\ \text{(EI)} \qquad \frac{(A\alpha)\phi}{\phi'}\ \text{(UI)} \qquad \frac{\phi'}{(E\alpha)\phi}\ \text{(EG)}$$

where ϕ' is the result of replacing all free occurrences of α in ϕ by some term in such a way that this new term does not become bound by any quantifier in ϕ'. Furthermore, in the case of EI, the substituted term must be a variable which is entirely new to the proof as thus far constructed. The new rule for boxing and cancelling is:

If the proof has a subpart which looks like

```
Show (Aα)ø
   X1
   .
   .
   .
   Xn
```

and (a) there are no uncancelled "Show" among X1...Xn, and (b) ø occurs unboxed among X1...Xn, and (c) α does not occur free in any line antecedent to this "show" line, then

```
*Show (Aα)ø
| X1
| .
| .
| .
| Xn
```

may occur as the next step of the proof.

I close this section with some short proofs to give a feeling for how theorems might be proved using this system, followed by the proofs of the same theorems in a resolution system. First, the theorem ((p→q)→(¬q→¬p)), which was the longest proof completed by the Logic Theorist (to be described in the next chapter).

```
1.      *Show ((p→q) → (¬q→¬p))
2.      |   (p→q)          Assumption
3.      |   *Show (¬q→¬p)
4.      |   |   ¬q         Assumption
5.      |   |   ¬p         2,4MT
```

Second, the theorem (p → ¬¬¬p), of which one can prove that the Logic Theorist's heuristics are incapable of proving. the Logic Theorist cannot prove it.

```
1.      *Show (p + ¬¬¬p)
2.        | ¬(p + ¬¬¬p)          Assumption
3.        |   *Show ¬p
4.        |     | p              Assumption
5.        |     | (p + ¬¬¬p)     4,Add
6.        |     ¬(p + ¬¬¬p)      2,R
7.        | ¬¬¬p                 3,DN
8.        | (p + ¬¬¬p)           7,Add
```

And finally a simple problem involving quantifiers.

```
1.      *Show ((Ax)(Px→Qx) → ((Ax)Px→(Ax)Qx))
2.        | (Ax)(Px→Qx)                Assumption
3.        |   *Show ((Ax)Px→(Ax)Qx)
4.        |     | (Ax)Px               Assumption
5.        |     |  *Show (Ax)Qx
6.        |     |   | Px               4,UI
7.        |     |   | (Px→Qx)          2,UI
8.        |     |   | Qx               6,7,MP
```

The proofs of these same theorems in a resolution system would go like this:

```
To prove ((p→q) → (¬q→¬p)):
1.   (¬p + q)      --clause from negation of conclusion
2.   ¬q            --clause from negation of conclusion
3.   p             --clause from negation of conclusion
4.   q             --resolve 1 and 3
5.   ☒             --resolve 2 and 4, yielding null resolvent


To prove (p + ¬¬¬p):
1.   ¬p            --clause from negation of conclusion
2.   p             --clause from negation of conclusion
3.   ☒             --resolve 1 and 2, yielding null resolvent


To prove ((Ax)(Px→Qx) → ((Ax)Px→(Ax)Qx)):
1.   (¬Px + Qx)          --clause from negation of conclusion
2.   Px                  --clause from negation of conclusion
3.   ¬Qa                 --clause from negation of conclusion
4.   Qx                   --resolve 1 and 2
5.   ☒                   --resolve 3 and 4
```

C. The "Axiomatic vs. Semantic vs. Natural Deduction" Division

The division in the previous section of systems of logic into axiomatic, semantic, and natural deduction was done as a matter of classificatory convenience. However, it is also a classification with which I have some considerable sympathy. Indeed, I will later argue that it is natural deduction to which we should turn our attention. However, I also recognize that the classification is not completely clear and the reasons I will adduce for restricting our attention do not apply with equal force to all versions of systems within each classification. This section therefore is an attempt to discuss the classificatory problems.

First let me say that I think there are two distinct types of reasons one could appeal to in preferring natural deduction to the other types of systems, and that I do use both types of reason. One has to do with a desire to try to follow the mode of reasoning employed by practitioners of logic and mathematics, while the other has to do with a theoretical appeal to "what logic really is". So far as the first reason goes, we wish to divide methods of proof into those which are (or: can easily be) employed by ordinary practitioners of logic. This division, as I see it, places the axiomatic systems and various of the semantic systems (especially resolution-based systems) on the side of "impractical", and natural deduction and various of the semantic systems (e.g., Jeffrey's tree method) on the side

of "easy to use". The second reason, I claim, places various of the semantic systems (e.g., Jeffrey's system, truth table lookup, and semantic tableaux methods) on the side of "not really logic", and natural deduction, axiomatic systems, and various semantic methods (e.g., resolution based systems) on the side of "true logic". If this is correct, it is seen that only natural deduction systems meet both of the desirability conditions.

Although it will be discussed again later, I remark now that the first reason -- ease of use by people -- clearly does classify the systems as I mentioned. Axiomatic systems and resolution systems are *very* difficult for most people, even accomplished logicians, to use; the natural deduction systems and the other semantic methods are recognized as being more akin to ordinary use. It is the second reason, and how it classifies, that I wish to discuss in this section.

On an intuitive level, the "axiomatic vs. semantic vs. natural deduction" division seems well-motivated and clear. But when one tries to give criteria which will unambiguously classify an arbitrary system of logic, the clearness of the divisions begins to vanish. But even though it be difficult to state the criteria, it seems to me that the division has sufficient intuitive pull that it should be kept. To that end then, I shall state what I take to be some of the hallmarks of each type of system, trying to justify placing the various systems discussed in the last section into the

categories I did. Simultaneously, however, I will present counterevidence to the adequacy of the proffered criteria, thereby showing that the different types of systems seem to collapse into one another.

So on the one hand it seems clear that there are (at least) these three different types of systems, on the other hand not much of methodological importance can be generated from the distinctions until and unless better criteria are discovered. At the end I will make some highly abstract remarks about what I take the *point* of logic to be. Given the view just espoused, that nothing of methodological significance is forthcoming from the distinctions, these abstract remarks are somewhat of an anomaly. For I propose to argue against pursuing axiomatic and semantic systems (in Chapter IV); and this argument is based on my comments concerning the point of logic and how it relates to the three types of systems.

One way of classifying systems of logic is according to whether they admit any formulae to be "not in need of proof", i.e., by whether they admit axioms. This seemingly clear-cut criterion would separate axiomatic systems from the others (semantic and natural deduction). However, even this apparently obvious distinction has been challenged. McCawley (1981: 44-46), for example, remarks that one could look at axioms as being a special kind of rule of inference -- namely the kind which have no premises. So there would be no difference in kind between the three-premise rule of

inference

$(P \rightarrow Q), (P \rightarrow R), (Q \rightarrow R) \vdash R$

and the two-premise rule of inference

$(P \rightarrow Q), P \vdash Q$

and the one-premise rule of inference

$P \vdash (P \rightarrow Q)$

and the zero-premise rule of inference

$\vdash P \rightarrow (Q \rightarrow P)$

An "axiomatic system" of the sort discussed in the last section happens to have one two-premise rule of inference (MP), one one-premise rule of inference (Sub), and two zero-premise rules of inference.

Although this is quite clearly correct, it seems to me that there is nevertheless considerable value in retaining a distinction between axiomatic systems and the others. Mostly this has to do with ease of psychological processing of proofs, but this is not the place to discuss such matters. It is perhaps also worth noting that the other systems might be considered converted to axiomatic systems by replacing rules of inference in them by axioms. For example the rule

$(P \rightarrow Q), \neg Q \vdash \neg P$

might be replaced by the axiom[7]

$\vdash (p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)$

So, while the axiomatic/non-axiomatic distinction is

---

[7]Of course there still have to be *some* rules of inference (as Lewis Carroll's tale of Achilles and the tortoise teaches us). Here I have in mind retaining MP and Sub. Furthermore, any rule which violates the Deduction Theorem must be retained as a rule, not as an axiom. (See the section below on the Deduction Theorem).

difficult to maintain theoretically, it seems clear that there *is* such a distinction.

More vexed is the distinction between semantic and natural deduction systems. It is perhaps instructive to begin by giving two clear reasons one might call a system semantic, and what the intent is in calling a system a natural deduction system. We shall then see that it is easy to construct a seemingly unbroken continuum between the two. While this might cause one to doubt the legitimacy of the distinction, I shall point to the clear instances of each in my attempt to justify classifying Kalish & Montague as natural deduction and resolution systems as semantic.

One clear sense of "semantic system" occurs when the prover refers to some representation of an instance of what is under debate; and, referring to the instance, makes claims about it and generalizes to obtain a proof of the theorem. I have in mind here such programs as geometry provers which, when constructing an argument about (say) equilateral triangles in general, actually construct a representation of a particular equilateral triangle, prove claims about it, and make various constructions within it. Having proved the relevant claim about the particular triangle, the prover generalizes to all equilateral triangles.

Another clear sense of "semantic system" occurs when the rules of inference each perfectly mirrors the intended semantic interpretation. Thus, for example, in a

propositional logic system where the semantic interpretation is just the truth table analysis of the formula, a system whose "rules of inference" were to construct truth tables and analyze them would certainly be considered "semantic". However, this is a clear case. One can deviate from the clear case and still be convinced one has a semantic system. For example, if the above system were to first negate the formula and do a truth table analysis on this negation, and claim it to be a theorem if the negation were uniformly false, this would be a clearly semantic system. A little further down this continuum are systems which use various shortcut methods on the truth tables: e.g., they use methods of evaluation such as "instead of looking at the truth or falsity of a conditional ($P \rightarrow Q$), look rather at the truth or falsity of $\neg P$ and the truth or falsity of Q". There are a variety of such "shortcuts" available. Indeed, Jeffrey's system of the last section has one such shortcut for each connective and its negation. For this reason it is properly called "semantic".

The hallmark of natural deduction systems is that the rules of such systems are supposed to mirror actual (valid) reasoning patterns which people employ. In general these patterns of valid reasoning are taken to be "universal" in scope and to apply to any subject matter. For that reason there is an attempt to view the rules as being completely separate from any semantic interpretation which might be placed upon them. In this respect, natural deduction systems

are close to the usual understanding of axiomatic systems (one is to ignore any interpretation which might be placed on these systems), and distinct from the central point of semantic systems (where one is constantly to make reference to the intended interpretation of (say) the premises of the argument under consideration).

In spite of the fact that the semantic/natural deduction distinction seems clear when one focuses on the characterizations offered in the last few paragraphs, it *is* merely a matter of focus. Thus, if one focuses on the question "Why is this step in this proof valid?", one will get an answer like the following from someone who uses a semantic system: "When you attend to what the objects under consideration are (the interpretation of the symbols), you will be immediately forced to acknowledge that the step *must* follow." Now contrast this with the natural deduction answer: "The step is valid because it follows a universally valid pattern of thought." One is tempted to ask, "Why do you think this pattern valid?" And the answer will inevitably be some variant of "You can't imagine a situation in which it would fail," or equivalently, "Given the interpretation placed on the symbols, it could not be otherwise." Here we see the two views of logic being forced to the side of semantic interpretation when the focus is upon the *justification* of the rules of inference.

However, one might wish to look upon the issue from a different point of view -- that of matching abstract

patterns. According to this way of focusing on the issue, we are not looking at the interpretation of the symbols, but rather at whether the system allows us to view a derivation as a series of steps each of which obeys a certain statable pattern. Looking at the matter this way -- whether the system is a matter of "symbol rearrangement" -- the answer (for most systems anyway) is yes. Even so semantic a system as Jeffrey's, with its exact mirroring of each type of truth table, would be a system in which one looks to whether formulae manifest certain patterns. Perhaps on this view, only systems that explicitly look to something besides the formulae would be semantic. (The geometry example and the explicit truth table analysis example might still be called semantic).

Yet there seems a clear sense in which (say) Jeffrey's system *is* semantic. It seems to me to be a matter of whether the rules of inference in the system closely follow the intended semantic interpretation versus whether they are taken to be independently statable. And if this be right, then there is a continuum along which different systems can be placed, from "clearly semantic" to "clearly natural deduction".*

So, given a system which is not at one extreme or the other, what grounds can be given for calling it "semantic"

---

*Looked at in this way, the axiomatic systems can be put anywhere along the continuum, depending on the extent to which the axioms and rules of inference follow the semantic interpretation. Most examples, however, are closer to the pattern matching end. (Such as the system mentioned in the last section).

or "natural deduction"? It seems to me that this is a matter of taste, but that there are certain guidelines. One guideline must be whether the patterns exemplified in its rules correspond to independently given, psychologically plausible modes of reasoning. Another guideline is whether the rules of inference closely follow the semantic interpretation in a step by step manner. Doubtless one could find others. The most contentious classificatory decision I made in the last section was probably placing resolution systems into the semantic category. But according to the two guidelines just given, that is exactly where it goes. The rule of resolution is certainly not psychologically plausible from any independent point of view. (If it were, it would have been discussed by some logicians before 1960. All the usual natural deduction rules were discussed by the ancient Greeks, the mediaeval grammarians, Boole, de Morgan, Frege, Russell, etc.) And while the resolution systems do not *precisely* mirror the semantic interpretation to the same degree as Jeffrey's system, or semantic tableaux, there is still a close correspondence: one is trying to see whether there is a possible counterexample. One does this by constructing formulae representing what such counterexamples would look like, interpreting quantifiers as describing certain entities in the domain, and the like.

I will now state dogmatically that I think "true logic" to be a matter of applying general reasoning patterns, patterns which are universal and apply to any subject

matter, to some specific problem at hand. That is, it is only those systems on the natural deduction end of our continuum that I take to be "really logic." The semantic systems are no more than a way of encoding facts about the specific problem at hand and applying some tools that work for [redacted] interpretation. And this is so even with "universal" semantic systems such as resolution-based systems. Even though resolution systems can be applied to any domain or interpretation, they amount to just a clever way of restating the interpretation and working on this restatement (as Jeffrey's system is just a clever way of restating truth tables and evaluating them). There are doubtless uses for semantic systems, but the study of logic and logical theory is not one of them, especially if one is concerned to describe human reasoning.'

D. Arguments and Theorems

It will not have escaped notice that I have been somewhat inconsistent in my use of 'theorem' and 'theorem prover.' On the one hand, in logic a theorem is a formula which can be proved in a certain way (namely, from no premises), while on the other hand the examples given in Chapter I of "theorems" would have it that a theorem is what follows logically from certain other given formulae (the premises). More mysterious, perhaps, is the use of 'theorem'

'In Chapter IV I again consider these types of systems and give further reasons for my choice of a natural deduction system as what I wish to investigate.

in axiomatic systems to describe what follows from the axioms and no other premises. Let us therefore be a little more precise here, but recognize that the literature on mechanical deduction is itself somewhat ambiguous on this terminological issue, and that I intend to maintain this ambiguity.

Let us start with natural deduction systems. One of the rules of proof in Kalish & Montague's system is that a premise can be introduced anywhere in a proof. If this rule is not employed anywhere in a given proof (as for example it was not used in the examples given at the end of the last section), then the formula being proved is a theorem in the technical sense. If the rule was employed, then the formula being proved depends upon, as it were, the premise(s) used. Semantically speaking, the formula proved has merely been shown "true *if* the premise is true"; whereas if the rule had not been employed then the formula proved is true (*simpliciter*). Now, had the premise itself been a theorem (in the technical sense) then of course the proved formula would also be; and if one did not know whether the premise were a theorem, one might say (in a somewhat loose sense) that the proved formula *follows as a theorem from* the premise. I suspect it is this sort of loose usage that has been picked up in the automatic theorem proving literature and which leads to the ambiguity noted above. A similar situation occurs in the semantic proof procedures. I noted in the last section that to prove a theorem (in the

technical sense) in Jeffrey's system, one negates it, roots a tree with this negation, and applies the various branching rules. One can introduce premises into this scheme by merely adding them to the root node and have the branching rules apply to all the formulae. Again one wants to say that if the premises were added to the root (and all branches closed) then the conclusion *follows as a theorem from* the premises; but if no premises were used then the proved formula simply *is* a theorem (technical sense). Similarly, in a resolution system if clauses corresponding to premises are used, then (if the null resolvent is reached) the proved formula follows as a theorem from the premises. In the axiomatic system exhibited in the last section, it is important to note that the propositional letters used in the axioms are propositional *variables* and can have as substitution instances any formula with (or without) such variables. The formula actually proved there, $(p \rightarrow p)$, was itself composed exclusively of propositional variables. Anything which can be proved from those axioms and the rules of inference is a theorem (technical sense). But premises contain propositional *constants*, i.e., the propositional letters in them are, semantically speaking, *interpreted*. Hence substitution is not allowed on them. One extends the definition of proof so that a premise can be entered as any line, but no propositional constant can have the rule of substitution used on it. The last line of such a proof will be a formula that, again, follows as a theorem from the

premises.

If one wishes to be more technical, one should use 'theorem' in the sense described as "technical" in the last few paragraphs, and use 'is a valid conclusion from the premises' for the sense described here as "loose". I shall, however, not be technical, except in cases where the context does not completely determine which sense is at issue.

E. The Deduction Theorem

When one extends the notion of theorem proving to cover arguments with premises, one must investigate whether there are any significant differences between "pure" arguments (without premises) and "applied" arguments (with premises). If we restrict our attention to arguments with only finitely many premises (which we shall throughout this work), then any differences will come out in whether the Deduction Theorem holds or not. Let $\Delta \vdash \phi$ stand for "(formula) ϕ is a valid conclusion from (the set of formulae) Δ"; intuitively, Δ is the set of premises and ϕ the conclusion. The Deduction Theorem states

if $\Delta \cup \{\psi\} \vdash \phi$ then $\Delta \vdash (\psi \rightarrow \phi)$

And since Δ is finite, repeated applications of the Deduction Theorem will eventually yield

if $\Delta \cup \{\psi\} \vdash \phi$ then $\vdash (\psi_1 \rightarrow (\psi_2 \rightarrow (\ldots \rightarrow (\psi \rightarrow \phi)\ldots))$

(where ψ_1, ψ_2,... are all the formulae in Δ). Since in all the systems under consideration

$(\phi_1 \rightarrow (\phi_2 \rightarrow \phi_3))$

and

$((\phi_1 \& \phi_2) \rightarrow \phi_3)$

are equivalent, we can more perspicuously state the result as

if $\Delta \cup \{\psi\} \vdash \phi$ then $\vdash((\psi_1 \& \ldots \& \psi) \rightarrow \phi)$

That is, for every argument there is a theorem corresponding to it which has the above form. However, in some of the systems under consideration here, the Deduction Theorem does not hold in general. The following is a valid argument in Kalish & Montague and in resolution systems

$Py \vdash (Ax)Px$

but

$\vdash (Py \rightarrow (Ax)Px)$

is not a theorem in either. In Jeffrey's system the former is not a valid argument and the latter is not a theorem; the Deduction Theorem holds in this system. The reason it does not hold in Kalish & Montague and in resolution systems is their handling of formulae with free variables. When they occur as premises or as conclusions in the Kalish & Montague system, or anywhere in a resolution system, they are implicitly universally quantified. Thus in these systems the above argument is equivalent to

$(Ay)Py \vdash (Ax)Px$

and the formula is equivalent to

$(Ay)(Fy \rightarrow (Ax)Fx)$

It is obvious now that the argument is valid and the formula is not a theorem. In Jeffrey's system, free variables are

taken to be like constants, thus explaining why neither is the argument valid nor the formula a theorem.

This sort of example shows that there really is an important difference between theoremhood and valid arguments, even when we restrict our attention to arguments with finitely many premises. We should make sure, in evaluating automatic theorem provers, that they are able to handle either failure or success of the Deduction Theorem, depending upon which underlying system of logic is being modelled.

III. A BRIEF SURVEY OF AUTOMATIC THEOREM PROVING: METHODS AND STRATEGIES

A. Soundness, Completeness, and Decidability

A system of logic, it will be recalled from Chapter II, is a recursively specified set (possibly empty) of axioms and a recursively specified set of rules of inference. Together with a definition of proof, these specify a set of theorems of the logic; and so in some sense one can identify the logic with its set of theorems. However, as we have already seen, there is another sense in which the logic is not the set of theorems because there may be a difference in what arguments two systems consider valid even though the theorems are the same. Since our interests include arguments in general, I will take the former characterization of a logic as the object under discussion, rather than the "set of theorems" characterization.

It is also my intention to take standard notions of "semantics" as given. Thus an interpretation of the logic will be a model structure having only well-known properties. A proposition (0-place predicate) has as value either **true** or **false**, an n-place atomic predicate is assigned an ordered n-tuple of items from the domain, a constant has as value an element of the domain, an n-place function symbol is assigned a set of ordered (n+1)-tuples of elements from the domain in such a way that whenever $\langle\alpha_1,\alpha_2,\ldots,\alpha_n,\alpha\rangle$ and $\langle\alpha_1,\alpha_2,\ldots,\alpha_n,\beta\rangle$ are in this set then $\alpha=\beta$. The propositional

connectives are given their usual truth functional interpretations, and the quantifiers are given an objectual interpretation. Having stated such an interpretation (of the predicates, constants, functions), one goes on to define what it is for a formula to be true under that interpretation. We shall employ standard definitions in this regard: An *argument is (semantically) valid* just in case: in every interpretation where all the premises are true so is the conclusion.

For an abstract system of logic to be sound, every argument one can construct a proof for is (semantically) valid. For it to be complete, every (semantically) valid argument must have some proof. Every system considered in Chapter II is both sound and complete.[10] (Of course since Jeffrey's system has different provable arguments than, say, Kalish & Montague's, this entails that they will have different semantics.) Since one has some particular understanding of the symbols in mind (a semantics), and one is using the logic as a way of manipulating the symbols so as to aid in characterizing this understanding, it is clear that one wants the abstract system of logic to be sound and complete. For if not, there would either be (semantically) "good" arguments that were not correctly so-called (by the

---

[10] Chapter II was purposefully vague with respect to which quantifier rules will be added to the "semantically-based" systems, and the axiomatic system was left unspecified beyond the implicational fragment. What is meant here is that there are straightforward completions (axioms, rules of inference) for all these systems which will render them sound and complete.

syntax), or else there would be (semantically) "bad" arguments decreed to be "good" (by the syntax).

What has thus far been said applies to the abstract systems of logic only. When these systems are implemented as a computer program, various other matters come into play in considering issues such as completeness. For example, any implementation of the axiomatic system discussed in Chapter II which was organized in such a way that (a) the theorem to be proved had to be a substitution instance of the consequent of an axiom and (b) the antecedent of that axiom had to be a substitution instance of an axiom, would obviously not be able to prove all theorems of the propositional logic. And this is so in spite of the fact that the system as a whole (which, as I have mentioned, is complete) allows the axioms, substitution in axioms, and modus ponens -- generally, all that is required to be complete. The idea is that a system's being complete means that *some* proof invoking the axioms, substitution, and modus ponens exists. But the *search organization of the implementation* -- while it is an implementation of a complete system -- may not itself be complete.

We shall see below similar cases. Indeed, we can even interleave another level of completeness here. The resolution inference system of Chapter II is a complete (abstract) system. Thus there is *some* series of resolutions which yield ⊠ for every (semantically) valid argument. Often one can prove that there is some specific way of organizing

these resolutions such that every proof exhibits that organization. I.e., one might say that every valid deduction can be given in a normal form. If so, then one says that this subsystem (the normalized forms) is itself complete and thus is equivalent to the entire system.

As before, we can talk about the search organization of the implementation for this normalized system, and find it complete or not. Unless one uses exhaustive searches for one's implementation, it is extremely difficult to achieve a complete search organization implementation. This is so because the abstract system's being complete means merely that there is *some* proof or other; the normalized system's completeness also only means that there is *some* proof or other obeying these constraints. But for a search organization implementation to be complete would mean that the method used will actually find the proof, given sufficient time to look.

Limitations to complete implementation come from two directions. The first is logical. It is a well-known fact that the first order predicate logic is undecidable, even though (abstractly) complete. What this means is that *as a matter of logic* there are formulae such that no matter how long one has been constructing a proof by a complete method, there will be no termination and one does not know whether the formula is provable (but the proof not yet finished) or whether the formula is not provable. So as a matter of logic there can be no complete implementation for the first order

predicate logic: even in the limit we cannot discover, for all formulae, whether they are or aren't theorems. The second direction of limitations comes from the inherent finiteness of machine implementations as opposed to the inherent infiniteness of abstract systems of logic. As one can see from the definitions of the vocabulary, terms, and formulae in Chapter II, the (abstract) logic we are interested in has an infinite number of variables, terms, and formulae. Yet by its very nature, an implementation is limited -- for example in the size of a formula which can be stored for consideration. Not only this, but also (in the abstract system) a proof can grow without end and yet be a correct proof. Thus there will be proofs which cannot be implemented, even if we are talking about a logic which is decidable and for which we have an otherwise complete implementation. So, although our system of logic may be complete (as is the first order predicate logic), and even though our normalized form of proof may be complete (as is the linear resolution we shall consider shortly), we may never be able to find a proof of a given theorem. I shall not consider, in this thesis, the problems of finiteness of implementation. For example, the system proffered in Chapters V, VI, and VII has only 40 variables at its disposal; so any theorem whose proof requires 41 variables will not be found, and the method will fail. Yet, since memory and size constraints are continually being revised upward, I do not consider this to be a crucial limitation.

More crucial limitations are (a) those that do not employ a complete normalized form (e.g., unit resolution, which we shall consider shortly), and (b) those which while utilizing a complete normalized form of proof, do not give sufficient search direction to find a proof.

This latter is a matter of degree. Thus linear resolution is complete; and systems utilizing it employ various strategies which "help proofs get started in the right way". Such strategies or heuristics are to be judged by how often they work. As we shall see, resolution-based implementations are particularly weak in this regard.

B. Early Heuristic Approaches

Computerized theorem proving begins with Newell *et al* (1957). They attempted to use the formalism of Whitehead & Russell (1910-1912) for the propositional calculus and to prove theorems of this theory. They claimed interest in human problem solving techniques, however; so they devised a proof search organization which would, they hoped, mirror that of humans attempting this same task. In particular they eschewed (what they called) the British Museum Algorithm of enumerating all possible proofs until the one being searched for appears. Of the various proof search organization techniques they employed, perhaps the most interesting one was that of "working backwards" from the goal to be proved to a "subproblem" that is perhaps simpler than the original goal. The techniques that they employed were called (by

them) "heuristics", a term that apparently means "intelligent guesses as to what to do next that have no assurance of working." Their success at proving theorems of the propositional calculus was very modest: the previous chapter shows what the hardest one they solved was and gives an example of a theorem which their "heuristics" are incapable of solving.

Gelernter's (1963) Geometry Theorem prover was similarly organized, although it employed a method of looking at a "diagram" which was offered with the statement of the problem. It too looked "backward from the goal", but managed to cut down on the number of possible "ways to go backward" by incorporating this "domain specific knowledge" about what is true in the diagram. The method invoked in these two theorem provers is called *problem reduction format*, a concept to which we shall return shortly.

C. The Decline of Heuristics

The 1960's saw a move away from this "heuristic" method of constructing proofs. Wang (1963) gave an algorithmic development of the propositional logic, with an extension to the predicate logic, using the methods of Herbrand, and argued strongly against the use of methods which were known not to automatically succeed ("heuristics") when there was an automatic method available (even if the automatic method was not how humans would solve the problems). We shall return to this issue in Chapter IV. This general *zeitgeist*,

coupled with the discovery of the resolution inference system (Robinson 1965, 1968), encouraged investigators to look for methods which were far removed from anything an actual logician would do to prove some alleged theorem. Since the system had a single inference rule (and associated substitution mechanism), it was felt that there was a sense in which this would be the simplest system of logic. Although elegant in this sense, the resolution procedure also produces intermediate clauses (thus generating subproblems) at an exponentially explosive rate.

A straightforward way of carrying out resolution on a set of clauses S is to compute all resolvents on pairs of clauses in S, add these resolvents to S, compute all further resolvents, add them to S, etc., until either the null clause appears or there are no further resolvents. That is, generate sequences S_0, S_1, S_2,... (levels of resolution) where

$S_0 = S$

$S_n = \{\text{resolvents of } C_1 \,\&\, C_2 \mid C_1 \in (S_0 \cup \cdots \cup S_{n-1}),\ C_2 \in S_{n-1}\}$

This procedure is the *level-saturation method*. Chang & Lee (1973: 92-93) carry out a simple example where $S = \{p+q, \neg p+q, p+\neg q, \neg p+\neg q\}$ and show that even in this simple case 34 further clauses (from two new levels) are generated before ⊠ is reached. Inspection of these new clauses reveals that there are a large number of superfluous clauses falling into three types. First, some of these clauses are tautologies. These cannot lead to ⊠ since tautologies are true under any

interpretation: if S is unsatisfiable, then the result of deleting a tautology from S will still be unsatisfiable. (Simple deletion is not quite all there is to the issue, however. If S contains only tautologies, one would not wish to delete them all and be left with ☒!) Second, a number of the clauses produced are identical with already-produced clauses. Clearly, these should be deleted. Thirdly, and including the second as a special case, some clauses are entailed by an already-present clause. Again these should be deleted. For example, if P(x) is already present, then (P(a)+Q(a)) should be deleted, since P(x) *subsumes* (entails) this clause by the substitution of *a* for *x*. The *deletion strategy* is the deletion of any tautology and any subsumed clause.'[1] Employing this strategy, Chang & Lee (1973: 94-95) show that the above example produces only five new clauses before encountering ☒ as the first clause at the second level. A special case of this is to delete subsumed literals within a clause (*unify*). Such a strategy might usefully be called a *heuristic* in the sense that it applies organizational control to the level-saturation method. Every theorem prover surveyed in the recent literature employs some such heuristic.'[2] But even with this (admittedly primitive) heuristic, it has been found that clauses are produced at too rapid a pace to make proofs of even

---

'[1]This is actually a version of the Davis & Putnam (1960) "Tautology Rule".

'[2]With the exception of Boyer's (1971) "lock resolution" method, which will be discussed in the next section and again in Chapter IV.

moderately simple theorems computationally feasible. And this problem led investigators to look at the conditions under which the production of intermediate clauses would be kept to a minimum: i.e., to more powerful heuristics. It is to these strategies that we now turn.

D. The Re-emergence of Heuristics

Linear resolution (proposed independently by Loveland (1970) and Luckham (1970)) is another overall way to organize individual resolutions, akin to the level saturation method of the last section.[13] The general idea is to pick a starting resolution, and from there on make resolutions that use the most recent resolvent. The advantage of this organization is in its clarity of presentation and ease of stating which (class of) resolution(s) to perform next. It is perhaps surprising that linear resolution is complete, that is, if S is an unsatisfiable set of clauses then there is some series of resolutions such that each one operates on the resolvent of the previous resolution and the last clause is ☒. This is surprising because one offhand thinks that there must be cases where two "independent" series of resolutions are needed in order to generate clauses where, finally, there is a resolution which "brings the two independent series together." Such a situation happens often in natural deduction and axiomatic systems. The reason this is not so

---

[13]This entire section benefits considerably from Loveland (1978) and Chang & Lee (1973).

in resolution systems has to do with the fact that in any such apparent case, so long as the two alleged "independent" series eventually can be "brought together", there must be some atom in one which occurs negated in the other. But then this pair of complementary literals must have occurred in the formulae which *began* each of the allegedly independent series; and in such a case, there is a different derivation which will start with them and operate in accordance with linear resolution. However, a word of deflation is in order for anyone who sees linear resolution as a solution for how to organize a resolution procedure. The completeness theorem says only that there is *some* linear resolution available -- it does not say how to find it. Thus suppose S = {(R+T+U), (P+Q), ¬P, ¬Q}. The set S is unsatisfiable, but no linear resolution starting with (R+T+U) will succeed. Clearly what is needed here is some strategy to tell where to start the linear resolution, for instance here we note that (R+T+U) is not resolvable with anything. But such a strategy is not enough, since the breakdown might occur anywhere. A "second-level breakdown" occurs if ¬T is added to S and a "third-level breakdown" occurs if both ¬T and ¬U are added.

One straightforward strategy would be to delete any clause containing an atom that does not occur in another clause. Obviously such a clause can never lead to ☒ because that literal can never be resolved away. A slight generalization of this would be to delete clauses containing a literal which is not complemented in another clause. (This

is Davis & Putnam's (1960) "pure literal rule".) If this strategy is carried out iteratively, one arrives at a system called *graph resolution*.

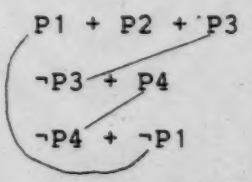
Kowalski (1974, 1978) provides the most fully described "connection graph" theorem prover. The idea behind a connection graph strategy is first, prior to trying to prove anything by resolution, to lay out the possible "resolvings out" as a graph. Thus suppose our clauses are

P1 + P2 + P3

¬P3 + P4

¬P4 + ¬P1

We draw connections between the literals which might resolve out, thus



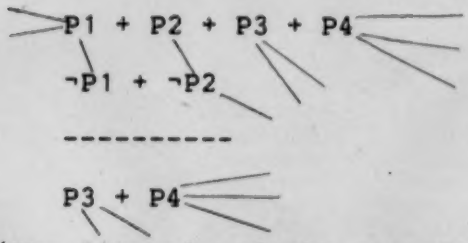
Any clause which contains a literal that is unconnected cannot possibly lead to the derivation of the null clause, and so is deleted, along with any of its connections. So in the above example we delete the first clause and its connections, leaving

¬P3 + P4

¬P4 + ¬P1

But these also now have unconnected literals and thus are to be deleted. If there were no other clauses, we would know prior to starting the resolution portion of an attempted proof, that it is not a theorem and hence not to start. Now

consider a slightly more complicated case. Suppose we have these clauses (amongst others -- their connections to the ones listed are indicated by lines to nowhere)



(The literals of the new clause introduced as a conclusion are connected to whatever P3 and P4 were connected to in the premise clause; and a connection, once resolved upon, is deleted). We now recheck for clauses containing unconnected literals. Using this technique, we can manage to cut down on many of the unwanted resolutions that would crop up.

One of the obvious strategies one might think of is called *unit preference*, and was introduced by Wos *et al* (1965).'' According to Loveland (1978: 98) it "is still not only the best known strategy, but...is one of the best refinements known to date." In order to deduce ☒ from a set of clauses, one must obtain successively shorter and shorter clauses. Unit resolution provides such a mechanism. The idea is to perform resolution operations where at least one parent clause is a unit clause (literal) before it would normally be performed under the search plan being employed. For example, if level saturation resolution is being employed, then prior to doing *all* the resolutions at a given level, first do the unit resolutions at that level and the

---

''Actually it is the "one literal rule" of Davis & Putnam (1960).

unit resolutions involving the new clauses thus generated, etc. Usually there is some mechanism to prevent creation of an infinite sequence of unit resolutions.'[15] For example Pf(x) and (¬P(x)+Pf(x)) unit resolve to obtain the unit clause Pf(f(x)) which starts such an infinite sequence.

Another obvious strategy to employ is *input resolution*. This is to make at least one parent clause of each resolution be an input clause, i.e., from S_0. The special appeal here is that the number of intermediate clauses will be kept small by this strategy, as an informal consideration of examples will show. Chang (1970) showed that unit resolution and input resolution are equivalent: there is a unit refutation from a set of clauses S if and only if there is an input refutation from S.

Unit refutation is not complete: not every unsatisfiable set will generate ☒ using only unit clauses, e.g., the example S = {(P+Q), (¬P+Q), (P+¬Q), (¬P+¬Q)} obviously has no unit refutation. Hence by Chang's theorem, it has no input refutation either. This raises the question of what is the class of unsatisfiable sets that *do* have a unit or input refutation? The answer is *Horn sets*. A Horn formula is a clause which has at most one positive (unnegated) literal, and a Horn set is a set of Horn formulae. This is an important set of formulae because many natural language "problems" can be stated as Horn

---

[15]Loveland 1978: 99 says to "arbitrarily choose" an upper bound on levels.

formulae.'‘ Even some other statements can be converted to Horn formulae by "consistent renaming of predicates" by their negations (see Meltzer 1966). If one clause has two positive literals, yet renaming is impossible, this formula can be converted into two Horn formulae by a "splitting rule", since the following is a tautology:

$$(S \& (A_1+A_2+C) \leftrightarrow (S\&(A_1+C)) + (S\&(A_2+C))$$

where: S is the conjunction of the other clauses of the set, (A_1+A_2+C) is the formula with two positive literals (A_1 and A_2) and a disjunction of negative literals C. If the set on the left side of $\leftrightarrow$ is unsatisfiable, then each of the disjuncts on the right side (each disjunct is not a Horn set) has a unit (or input) refutation. However, there may be shared variables between A_1, A_2, and C; and that means that it is possible that the variable instantiations used in showing $(S\&(A_1+C))$ unsatisfiable and showing $(S\&(A_2+C))$ unsatisfiable are incompatible and so there is no single instantiation which shows $(S\&(A_1+A_2+C))$ is unsatisfiable.'' What is needed is some method of showing the interaction of variables in the two sub-refutations. Typically, one processes one of the Horn formulae, and retains all the instantiations of variables in the second formula by the appropriate instantiation. (The method used is the same as

---

'‘The clause form of Horn formulae is $(\neg\phi_1+\neg\phi_2+\ldots+\phi)$, which is equivalent to $(\phi_1\rightarrow(\phi_2\rightarrow\ldots\rightarrow\phi))$ and to $((\phi_1\&\phi_2\&\ldots)\rightarrow\phi)$, a very common natural language form, corresponding as it does to a GPS-style problem setting and to natural language universal statements.

''That is, just because each of $S\&(A_1+C)$ and $S\&(A_2+C)$ has a refutation, it doesn't follow that $S\&(A_1+A_2+C)$ does.

"answer extraction" discussed in Chapter I).

Various other splitting techniques are available, but I shall not discuss them further. See Chang & Lee (1973), Henschen & Wos (1974), and Nevins (1974), all of which are refinements of Davis & Putnam (1960).

A "generalization" of the unit preference strategy, called the *fewest literal preference* strategy, was proposed by Slagle (1965). One does a "modified depth first" method where, when a pair of formulae are chosen to be resolved upon one continues to resolve upon that resolvent, its resolvents, etc., until some specified threshold number of levels has been reached. When this happens, pick another pair of formulae to be resolved upon in this manner, etc. (Of course, if ☒ is derived at any step, terminate with a proof). At each stage one wishes to process the "likely looking" candidates before the "bad" candidates. Since we are trying to reach ☒, the "most likely" candidates will be ones that result in the smallest resolvent (smallest in the sense of having fewest literals). A measure of this is the sum of the lengths of the parents. Hence, do resolution in this order. Obviously the method is complete, since eventually all resolvents will be computed.

Using length of parent clauses is only one of a number of possible measures of "goodness" of a proposed resolution. Among the other possibilities mentioned by Chang & Lee (1973: 154) for judging whether to resolve C with B are

1. The number of literals in C

2. The number of clauses which can resolve with C
3. The number of constants in C
4. The number of function symbols in C
5. The number of distinct variables in B and C
6. The number of constants in C + (1+the number of variables in C)
7. The number of literals in both B and C
8. The number of distinct predicate letters in B and C
9. The length of C + the length of B

Using standard methods from the analysis of variance, one estimates how "good" it is to resolve C against B by weighting each of the possibilities and summing them. Thus letting $f_1,...f_9$ stand for the nine listed *features* we might find important, the estimate h(C,B) of how good the resolution of C against B is, will be given by

$$h(C,B) = w_0 + w_1 f_1(C,B) + \ldots + w_9 f_9(C,B)$$

If we know a number of specific values of $h^*(C,B)$ -- how "good" in a number of different cases C against B resolution really turned out to be -- we can use standard regression analysis techniques to get values for $w_0, w_1, \ldots w_9$. This then raises the question, how do we know if a set of features is good? If the set is good, then after a number of examples used to obtain the *w*'s so as to get *h*, it should be useful in new cases. If it does not apply well to new cases, new features should be found. h(C,B) is called the *heuristic evaluation function*; see Slagle & Farrell (1971) for further discussion and examples.

One of the first and still most popular strategies is the *set of support* strategy of Wos, Robinson & Carson (1965). Recall that a deduction of C from premises P_1, P_2,... amounts to showing that $\{P_1, P_2, \ldots, \neg C\}$ is unsatisfiable. Since $\{P_1, P_2 \ldots\}$ is generally satisfiable, it is perhaps wise to avoid resolving their clauses. This is what the set of support strategy tries to accomplish. Given a set S of clauses, a subset T of S is called a set of support if (S-T) is satisfiable. A set of support resolution is a resolution such that not both parents come from (S-T), and a set of support deduction is one where every resolution is a set of support resolution. It is quite straightforward to show that this strategy is complete: if S is a (finite) set of unsatisfiable clauses and T a subset of S such that (S-T) is satisfiable, then there is a set of support deduction of ☒ from S with T as the set of support. Obviously there can be more than one set of support for an unsatisfiable S. In Chapter IV I consider some of the problems this observation raises for implementing set of support strategies in a reasonable way.

Hyperresolution was introduced by Robinson (1965). The idea behind it is that one can specify a *setting*'* to aid in the choice of which formulae to resolve against each other. A setting is intended to be a way of dividing all clauses into two classes: those true in the setting vs. those false

---

'*Or interpretation, model, partition, partial interpretation, etc. The literature is rife with terminology. I here follow Loveland (1978: 116).

in the setting. This is accomplished by making the setting be a consistent list of literals which occur in the clauses. (Since the set S was unsatisfiable, no setting can satisfy every member of S nor falsify every member of S; thus S is partitioned into two non-empty subsets). A resolution is performed only when the parents each come from different sides of the partition, and the resolvent is then placed in the appropriate side. Of special interest are the two settings:

{A|A is an atom in S}

{¬A|A is an atom in S}

called the positive and negative setting, respectively. Resolution using the negative setting is hyperresolution, that using the positive setting is called P_1-resolution.'' Chang & Lee (1973: 108) point out that in a very common case, the premises of an argument are represented by either positive or mixed clauses, while the negation of the conclusion is represented by a negative clause. In this case, hyperresolution is best viewed as "working forward" (from the premises) to deduce the original conclusion and thence resolve against its negation; while P_1-resolution is best viewed as as "working backward" from the negation of the conclusion to ☒. It should be noted here also that "consistent renaming of predicates" involving uniform

---

''Mysteriously, Chang & Lee (1973: 108) call the former positive hyperresolution and the latter negative hyperresolution. Their reason is that when the strategies are used, all resolvents of the former will be negation-free while in the latter they will be completely negative.

replacement of predicates by their negations (Meltzer 1966) can aid in finding reasonable settings. Meltzer considers some cases involving the number of positive and negative literals in a set of clauses S where this information is used to find good settings. (Meltzer calls this Pp-resolution).

The fact is that the set of support strategy, the P_1-resolution strategy, the Pp-resolution strategy, and the hyperresolution strategy are all special cases of a more general strategy: semantic resolution (Chang & Lee 1973, Chapter 6, who attribute it to Slagle 1967). It should be obvious that P_1-resolution, Pp-resolution, and hyperresolution differ only in what they choose as the setting. In fact semantic resolution amounts only to saying that *some* setting should be chosen and used to divide the clauses into two disjoint sets. Clearly the present types of resolution merely offer advice on how to choose a setting. The same is true with the set of support strategy. One constructs a setting, usually using the negation of the conclusion, and constructs the two sets C (the set of support) and (S-C). All resolutions are to come from parents in different sets. When a resolvent is found it goes into the (S-C) set and will be resolved against a member of the C set. Obviously this is again an example of the general semantic resolution. Semantic resolution is complete in the following sense: if S is unsatisfiable and I is any

setting[2•] then I partitions S into C and (S-C) and there is a deduction of ⊠ from S such that each resolution has a parent from each of C and (S-C). From this completeness theorem, the completeness of any of the special cases follows.

One final strategy should be mentioned here, as it is common in the literature and takes up considerable space in textbook presentations such as Chang & Lee (1973) and Loveland (1978), and that is *ordering*. Suppose we arbitrarily give an ordering to the predicate symbols occurring in a set of clauses S, and require that when two clauses are resolved, we always resolve upon the largest literal. Thus for example, if we have an ordering P>Q>R and the two clauses (P+R) and ¬R, we would *not* be allowed to resolve them because R is not the largest literal in the two clauses. One can add this sort of order-of-predicate strategy to either linear or semantic organizations of resolution and the resulting system is complete: from an unsatisfiable set of clauses there will be the appropriate (linear, semantic) ordered deduction of ⊠. Nonetheless, as mentioned before, such information does not uniquely determine which resolution to start with to even get a deduction of ⊠, much less an optimal one. All we know is that there is one, not where to start or how to proceed in order to find it.

[2•]Recall that settings must be consistent.

There is another refinement of ordering that should be mentioned, and that is to order individual *clauses* rather than just the predicates as a whole. This refinement is carried out differently in the semantic and linear systems. To order a clause, one merely uses the order of occurrence (left-to-right) of the predicates in the clause. (One could choose some other order, but this one is easy and illustrates all the points.) Now for some technical details. If two or more literals of an ordered clause C (with the same sign) have a most general unifier, substitute the unifier uniformly and delete all larger occurrences of the literal (i.e., those later in the clause) to obtain the *ordered factor of C*. If C_1 and C_2 are ordered clauses with no variables in common, and L_1 and L_2 are literals in C_1 and C_2 respectively, and L_1 and $\neg L_2$ have a most general unifier s, and if C is the ordered clause resulting by disjoining C_1s and C_2s, removing L_1s and L_2s and deleting any literal identical to a smaller (in C) literal, then C is an *ordered binary resolvent* of C_1 against C_2. (Note that this notion is not symmetric). Now, an *ordered resolvent* of C_1 against C_2 occurs when: (a) C_1 and C_2 are both ordered clauses, (b) C is an ordered binary resolvent of X against Y where X (and Y) is either C_1 (or C_2) or an ordered factor of C_1 (or C_2). Now suppose I is a setting. An *ordered semantic clash* with respect to I occurs when (a) there is a sequence of ordered clauses C_1, C_2, ..., Cn (b) C_1, C_2, ..., Cn are all false in I (c) For each i=1,...n there is an ordered resolvent R_{i+1}

of Ci against Ri (d) The literal in Ci that is resolved upon is the last literal in Ci, and the literal resolved upon in Ri is the largest literal that has an instance true in I, and (e) R_{i+1} is false in I. An example would perhaps help. Chang & Lee (1973: 115) give this: Let S be the set of ordered clauses {(Q(a)+R(x)), (¬Q(x)+R(x)), (¬R(x)+¬S(a)), S(x)}. Let I be an interpretation where every literal is negative. Then the following is an ordered semantic deduction.

(Q(a)+R(x)) S(x) (¬R(a)+¬S(a)) yields Q(a)

Q(a) (¬Q(x)+R(x)) yields R(a)

R(a) S(x) (¬R(a)+¬S(a)) yields ☒

Slagle and Norton (1971) experimented with ordered clause, semantic resolution. Unfortunately, it is not complete. The following set of formulae are inconsistent but there is no ordered clause, semantic refutation of them: {(P+Q), (Q+R), (R+W), (¬R+¬P), (¬W+¬Q), (¬Q+¬R)}.[21]

A somewhat different refinement of ordered clause, semantic resolution is Boyer's (1971) *lock resolution* where each occurrence of a literal is assigned an integer index; different occurrences of the same literal in the set of clauses may be indexed differently. Resolution is then permitted only on literals of the lowest index in each clause. The literals in the resolvents inherit their indices from their parent clauses; if there is more than one possible index assigned this way, it is given the lowest

---

[21]The example is due to Anderson (1971).

index. This kind of ordered clause, semantic resolution is complete: there is some lock resolution of ⊠ from S, if S is unsatisfiable. But there are other problems with it which will be discussed in Chapter IV.

It was mentioned before that input resolution is not complete: there are proofs in which some resolutions have to be made between two clauses which are each themselves resolvents of other clauses. It would be nice to be able to identify such resolvents. Loveland (1968) introduced the *model elimination strategy* for this. Basically the idea is to record which literals have been resolved upon and therefore deleted; in fact, if we have determined that we must use some previously generated resolvent -- we need not even remember which one -- we merely perform some operation on the clause to obtain a new clause. When two (ordered) clauses are resolved against one another, we keep the (unnegated) literal resolved upon in a special format, say as italicized. Such a literal is said to be *framed*. Thus in (P+Q) and (¬Q+R), both treated as ordered clauses where the first is resolved against the second, yields (P+*Q*+R). Some details now are that a framed literal which does not precede any unframed literals is deleted, that for multiple copies of the same framed literal we delete all but the left-most one, and that whenever an unframed literal of a clause is complementary to a framed literal, then frame that complementary literal. (Intuitively, this last occurs when some previously-generated resolvent will be needed to

resolve against the current one. Here we do not remember what one it was, only that there had to be one such.) So as an example (taken from Chang & Lee 1973: 137-138), consider S to consist of the ordered clauses {(P+Q), (P+¬Q), (¬P+Q), (¬P+¬Q)}. Starting with (P+Q), the last literal of this ordered clause is Q, which can be resolved against (P+¬Q). So, recording the relevant information, we get (P+P+*Q*); but then we delete duplicate occurrences of P, and note that *Q* is not followed by any unframed literal and so is deleted. Thus we get P. Its last literal is P, which can be resolved against (¬P+Q). The answer yielded here is (*P*+Q). Its last literal is Q, which can be resolved against (¬P+¬Q). The answer obtained is (*P*+*Q*+¬P). We now note that the last literal of this formula is the negation of one of the framed ones. We therefore frame it also, yielding (*P*+*Q*+¬*P*). But now there are no unframed literals after the framed ones, so we delete the framed literals, yielding ☒. It can be shown that if S is unsatisfiable, then there is a linear resolution using the model elimination ordering strategy which will generate ☒. One might also note that set of support is a special case of this strategy.

E. The Retreat from Resolution, I

The first thing one notes about resolution is the completely unnatural format required, clausal form. With the exception of those who write automatic theorem provers (and those people only while they are writing), no one uses

clause form to represent problems whether they be in mathematics, program specification, or even pure logic. Thus there is perceived a need to construct a resolution-based system which allows analogues of the resolution rule to operate on arbitrary formulae of first order logic. So there has been a search for a system with a single inference rule that can be used like this. Murray (1982) proposed what he calls NC-resolution, a procedure which is extraordinarily complex and involves reducing truth functional expressions to simpler ("reduced") ones, replacing sub-formulae by truth values, and finally unifying the formulae (after checking their "polarity" -- the number and position of negations in the subformulae) to arrive at a disjunction of the reduced formula. A proof is complete when F ("the false formula") is reached. The actual method used after the reduction is a "semantic resolution" of the sort described in the last section.

The 1970's saw more and more moves away from the pure resolution systems. A number of these newer approaches attempted to incorporate natural deduction techniques to divide "hard" problems up into several easier ones and then turn these easier ones over to a resolution prover. The Davis-Putnam (1960) "split" rule, discussed in the last section with reference to Horn clauses, is a primitive version of such a heuristic strategy. Two works should be singled out here. Nevins (1974) starts with an unsatisfiable set S of formulae and removes formulae from it one at a time

(complex formulae first), breaking them down by rules akin to Jeffrey's (1967) rules of Chapter II, and placing the results into set S_1. Thus if $\neg(P \rightarrow Q)$ is in S, it will get removed and the formulae P and $\neg Q$ will be put into S_1; if $\neg Q$ is in S_1 and $(P \rightarrow Q)$ is removed from S, then $(P \rightarrow Q)$ and $\neg P$ are added to S_1. When S_1 contains a formula and its negation, the proof is finished. Disjunctions are handled differently by being split into subproblems. If $(P+Q+R+...)$ is in S then (a) if no variable in P subsumes any of Q, R,..., make a copy of S_1 and add P to it and start again. If successful, do the same with Q, then R, etc. If all of them yield a proof, then the set S is unsatisfiable. (b) If P does subsume one of the other formulae in the disjunction, a more complex plan is required. The mechanism proposed (pp. 609-611) attempts first to find other splits, and failing that attempts to keep track of all the variables as if they "depended upon" the disjunction being split. Nevins considers the effect of this method when another such disjunctive split is required and there are unifiers *within* both disjunctions and *between* the disjunctions. So far as I can see, his method cannot work in general. Nevins does not consider the method discussed in the last section concerning splitting "almost" Horn formulae.

Bledsoe (1971) describes a similar system. In this system a number of attempts are made to simplify and break up the to-be-proved theorem before the results are sent on to a resolution subroutine. Many of these correspond to the

tactics one would use in Kalish & Montague's system, e.g., if one wishes to show that (P&Q) was a theorem, one would spearately show that P was a theorem and then that Q was. Similarly, to show that (P↔Q) is a theorem, one shows that each of (P→Q) and (Q→P) is a theorem. Bledsoe's system even uses some strategies not immediately available in Kalish & Montague's system,[22] for instance to prove ((P+Q)→R) prove first each of (P→R) and (Q→R). It should be noted that these splitting strategies actually only apply to the theorem to be proved. Once the theorem has been "split" into the (hopefully) simpler subproblems, the resolution routine is called on them. A "tight timelimit" is kept on resolution, and if the proof is not forthcoming within the timelimit, a set of "actions" is taken. The actions to be taken depend upon the specific domain under study; for instance, in his (1971) the domain was set theory, so various set theoretic reductions were employed as "actions" (and called "reductions"). E.g., in certain (syntactically defined) formula-positions such as consequents of conditionals, the set equality symbol A=B was replaced by the inclusions (A⊂B) & (B⊂A), and sometimes the inclusion (A⊂B) was replaced by set membership (Ax)(x∈A→x∈B). Splitting is done again and the resolution program is recalled.

As I see it, this sort of thing -- non-clausal format, truth functional breakdown, splitting of the goal, and

---

[22]They are not available because they do not correspond to primitive rules of inference in their system. Kalish & Montague eventually introduce these as derived strategies.

domain-specific reduction -- is the first step away from resolution procedures.

F. The Retreat from Resolution, II

Further breaks from pure resolution were already underway, headed by Bledsoe and people surrounding him (to judge from the literature). In 1972 Bledsoe *et al* replaced the **resolve** subroutine of his 1971 system (discussed in the previous section) by one called **imply**. The authors claimed to "have thus eliminated resolution altogether from [their] program, replacing it by an 'implication method' which [they] believe is faster and easier to use...". As in the 1971 system, there are two parts to the system: the proof system proper and a variety of (domain-specific) routines. In the 1971 system these latter were called "reductions" (for set theory), whereas in the 1972 system they are a set of methods for simplifying and solving linear equations by assigning sets and types to them. One of these methods is called the "limit heuristic" -- whose application here is strictly restricted to proving limit theorems. But the strategy behind it is of some more general interest. In the words of the authors,

> Because the limit heuristic enables our program to prove many theorems about limits, we regard it as a rather interesting trick. But more interesting and important than the fact that it works some problems [sic] is the principle behind it. That principle

might be stated:

> To establish a conclusion C from several hypotheses, among which is H, *force* H to contribute all it can towards establishing C and leave a *remainder* to be established with the help of *other* hypotheses.
>
> ...[I]f one can truly make H contribute all it can towards C, then H is not needed to establish the remainder. That is, a reduction in the number of hypotheses is achieved while a significant step in the proof is made.

This guiding principle can be more widely applied than just in this limit heuristic, as the authors acknowledge; indeed it could be adapted to the proof system in general -- even to pure resolution, where it would be some kind of depth-first search. We shall return to the shortcomings of this general principle in the next chapter.

The proof system proper of (1972) contains two parts. The first is the same as the (1971) system, namely a set of "splitting" heuristics which break the main theorem to be proved into subgoals. The second is the **imply** routine to which these subgoals are now passed (instead of the **resolve** subroutine as before). This subroutine has two arguments: the formula C to be proved, and R (a "reserve" set of formulae). The result of a call to **imply** is either a substitution or **nil**. The latter indicates failure to establish the subgoal. **imply** attempts to find and return the

most general substitution S such that (R→C)S is true. If S is the empty substitution, then **imply** returns **true**.

A formula C is converted to a quantifier free "skolemized" form following the method of Wang (1963),[23] and then a call to (**imply** C **nil**) is made. Various of the parts of **imply** look at the form of C to decide what to do next. Some of these rules say to solve other problems. Thus if C was of the form (H→(A→B)), the relevant rule recursively calls (**imply** (H&A→B) **nil**). So for this example we now look to the relevant rule which says that if (**imply** (H→B) A) returns S_1, then (**imply** (H&A→B) **nil**) returns S_1, but if (**imply** (A→B) H) returns S_2, then (**imply** (H&A→B) **nil**) returns S_2. So we need to evaluate (**imply** (H→B) A). Assuming H and B to be positive and non-complex, this succeeds if either H is tautologically equivalent to B (and then we return **true**) or else there is a most general unifier of H and B (in which case we return it). Such a simple case would only happen when H itself directly implied B without the need for A. In any more complex case, say a case where H had the form (A→B), we will need another call to **imply**. So we are trying to evaluate (**imply** ((A→B)→B) A); the relevant rule is "backwards chaining". We need to evaluate (**imply** (B→B) A) -- which yields **true** -- and (**imply** (A→A) NIL).[24]

---

[23] This is not clause form since one does not convert first to prenex normal form.

[24] In the more general case, the consequent of this last formula will be the result of substituting the most general unifier found from the previous call to **imply**. In the current example the empty unifier was found, and so here we did no substitution. Strictly speaking, the statement of backwards chaining is: (**imply** ((A→B)→C) R) first calls

This last also yields **true** and is percolated backwards to the first call to **imply**.

A careful examination of the rules of **imply** (1972: 36) shows that, besides all the "rewriting" rules, there are the following proof strategies. First, conditionals are proved when a most general unifier of the antecedent and consequent can be found. Second, a reductio-style proof is attempted in the case where formulae of the form (H→¬C) are being proved. This is done by changing (**imply** (H→¬C) R) to (**imply** (H&C→**nil**) R). And finally there are the two strategies which call the "reserve" R into play. The first is the "backwards chaining" described above, and the second is when the formula to be proved is (¬H→C) with reserve R we call (**imply** (R→H+C) **nil**). An examination of proofs of any complexity at all will show that backwards chaining is the most commonly-attempted strategy, and that besides the finding of a unifier, it is what really drives **imply**. This raises the question of whether backwards chaining is a good choice of strategies to rely upon almost exclusively. I discuss this in the next chapter.

Bledsoe & Bruell (1974) is an interactive theorem prover, arranged along the same lines as the Bledsoe *et al* (1972) system. Its expert domain is topology, and so it has a number of "reduce" style techniques relevant to this area.

---

[2] (cont'd)(**imply** (B→C) R). If this returns S_1 it then calls (**imply** (R→AS_1) NIL). If this returns S_2, then the top-level call returns (S_1S_2+S_2) where neither S_1 nor S_2 are NIL. If the second call failed but either of (**imply** (R→AS_1+C) NIL) or (**imply** (((A→B)&R)→AS_1) NIL) yields S_2, then again the top-level call returns (S_1S_2+S_2).

While the main interest of the 1974 system is in the variety of ways a user can interrupt, add "axioms" to be used, suggest substitution instances, etc., there are some parts of it relevant to the current discussion of theorem proving strategies.

The main differences between the 1972 and 1974 systems are these. In the 1972 system, the **split** routine simplified the overall goals, but not any of the subgoals which might be set up after the proof has started. Similarly, the **reduce** routine was applied after the **imply** routine had failed, and then **imply** was recalled. All this was handled by an overall monitor called **cycle**. The current system makes **imply** be the overall monitor, and it calls the **split** and **reduce** routines as needed. One further kind of strategy was added to **imply**: ground forward chaining. Forward chaining was omitted earlier because it produces a large number of intermediate and useless clauses. A *ground* forward chain occurs when the expression being chained off of contains only constants. Furthermore, since this system now interacts with a human user, certain strategies can now be relegated exclusively to the user. Thus, there was before a "backtrack" strategy used when calling (**imply** (H→(A&B)) R), which tried (**imply** (H→A) R) and used the unifier S thereby found to try (**imply** (H→BS) R). But if this last failed, a backtrack was used to find another unifier S_1. In the current system, such failures are handled by the human, who suggests other unifiers. The current version now uses a "breadth first" search rather

than the earlier "depth first" search. And coupled with this is a routine which tries to use a hypothesis which is "like" the desired conclusion, even though a complete match (unification) cannot be made. As will be shown in the next chapter, it is not obvious that this is a wise change. It should finally be added here that the portion of the **reduce** strategy which substitutes definitions now has an ordering imposed on the terms (from "common" to "strange") so that "strange" terms get defined first and the **imply** routine tries to prove with this before attempting to prove after defining "common" terms.

IV. METHODOLOGICAL AND PRACTICAL PROBLEMS IN AUTOMATIC THEOREM PROVING

A. Introduction

As can be seen from the examples cited in Chapter I, the different uses to which automatic theorem proving can be put set different criteria upon the way the automatic theorem prover is to be organized. For example, Polly Programmer's verifier sets as a necessary condition that the proof checker should be sound, i.e., that if the verifier says the program is correct then it is correct. Polly might also wish to set the condition that the verifier be complete, i.e., that if the program is correct then the verifier will say so. So long as these conditions are met, Polly has no other cares about *how* the verifier carries out its task; natural deduction, resolution, and axiomatic systems never enter her mind. Larry Lazy, on the other hand, must guarantee that his automatic program synthesizer will construct the relevant proof in such a way that the parts of the proof correspond to standard programming constructs of PASCAL. He is therefore *not* permitted to use just any logical method. He might indeed be willing to give up completeness and even soundness if the system were to construct a correct PASCAL program in a large number of cases and which often gave "almost correct" programs. Felicity Findout requires a system which will allow her to obtain answers about *who* did such-and-such and *how* is

so-and-so series of actions to be performed. She thinks that the sort of systems initiated by Green (1969) most naturally allow for this sort of "answer extraction", and therefore has implemented a resolution system with answer extraction. Robbie Robot's actions are set up in such a way that certain subgoals need to be stated and proved along the way to performing the overarching task. It seems natural, in such a system, to use natural deduction techniques which explicitly state the subgoals as part of the proof, e.g., as done in the Kalish & Montague system. Finally, in the area of cognitive studies, which possibly includes natural language processing, one is obligated to try to mirror the actual processes an ordinary human goes through in attempting to solve intellectual questions. In such studies, one needs first to find out how real people do this in simple cases and build one's theory from there. Anecdotal evidence (of the sort alluded to in Chapter II) indicates that the majority of people perform logic tasks in some type of natural deduction system, often explicitly setting themselves subgoals.

In the field of automatic theorem proving the overwhelming movement has been the investigation of resolution systems (with ever more fancy control structures). So nearly universal has this movement been that even in the fields where it might not seem so natural to use resolution methods (e.g., automatic program synthesizing), the strongest force has been the attempt to invent

resolution strategies which will perform the job (see Bibel 1979 and Guiho & Greese 1980 for example).

The major emphasis of the present thesis is to construct an automatic theorem proving system using natural deduction techniques so as to investigate the intricacies of such systems. I shall attempt to show that natural deduction leads to straightforward implementation of easy-to-understand strategies which correspond naturally to human-like proof methods. The resulting proofs are not only solved in the same manner as people do, but the intermediately difficult and very difficult proofs are performed more efficiently than resolution systems do them. The conclusion I draw from such performance is that the use of human-like strategies is, in the long run, the best research direction for automatic theorem proving in the future. Instead of trying to force resolution procedures to account for those uses of automatic theorem proving where other methods seem more natural, perhaps the time has come to try to use natural deduction techniques in areas where resolution procedures have achieved some success. At the very least I hope to show that natural deduction techniques are easy to implement on a computer and that therefore one need not expend one's energy on trying to invent methods of constructing new resolution methods so that those areas where natural deduction is the obvious choice can be treated. Instead, one can merely adopt the sort of automatic theorem prover displayed in this thesis.

In general, my feelings about natural deduction systems vs. resolution systems follow those expressed by Bledsoe (1977). He thinks that "natural systems" are both easier for human use and for machine use of knowledge bases.[23] As far as "human use" goes, Bledsoe points out that with natural deduction systems one can bring mathematical knowledge (for example) to bear in the same form as it is used in mathematics, that the mathematician can easily recognize places where such knowledge can be used, that the systems are easier to design and work upon, and that such systems are essential for mathematician-machine interaction. He also finds them easier for machine use of knowledge along these dimensions: they automatically limit the search by not starting all proofs as a "syntactic search strategy" does; they are a natural vehicle upon which to hang heuristics, knowledge, and "semantic search strategies"; they make it easier to combine procedures with deduction; and they solve the contextual data base problem.[24]

---

[23]Bledsoe's terminology does not always follow standard usage. He includes both our natural deduction systems and the Logic Theorist in his "natural systems". He furthermore often calls such systems "semantic" and distinguishes them from those with "syntactic search strategies." The careful reader of the last two Chapters will find the Logic Theorist drastically different from natural deduction systems, similar only in that one can easily implement such strategies as chaining in both of them. Such a reader will also conclude that it is the resolution systems which are properly called "semantic". Indeed, the Logic Theorist and the natural deduction system soon to be described are as "syntactic" as could possibly be. And finally, given the discussion of the last chapter about the use of heuristics in resolution systems, such a reader would be chary of claims about the total unsuitability of resolution systems to incorporate "semantic search strategies."

[24]It is not clear how they do this last. Bledsoe (1977: 15)

Whatever one thinks about Bledsoe's specific terminology, there is clearly (or so it seems to me) a point to be made in favour of natural deduction along all these lines. And it is this feeling I intend to try to inculcate in the reader by my development (in the next few chapters) of a natural deduction system. Generally speaking, I intend to investigate how people perform the intellectual task of producing a proof, and "try to write a program which performs in the same way. In this I follow Bledsoe's lead, who says (1977: 2)

> The author was one of the researchers working on resolution type systems who "made the switch". It was on trying to prove a rather simple theorem in set theory by paramodulation and resolution, where the program was experiencing a great deal of difficulty that we became convinced that we were on the wrong track. The addition of a few semantically oriented rewrite rules and subgoaling procedures made the proof of this theorem, as well as similar theorems in elementary set theory, very easy for the computer. Put simply: the computer was not doing what the human would do in proving this theorem. When we instructed it to proceed in a "human like" way, it easily succeeded.

Indeed, my current work might be best viewed as trying to get a good version of Bledsoe's pure proof strategies

---

[3] (cont'd)claims only that resolution provers require one data base for each clause.

working. Only after this is done, and thoroughly tested on examples from pure logic, is it time to add further heuristics of the sort called domain-specific reductions by Bledsoe. I thus want to ensure that the domain-independent portion works as desired before adding anything relevant to domain-specific strategies.

B. Some Theoretical Remarks about Theorem Proving by Machine

It is not always clear what particular investigators want out of their theorem provers. One might, for example, wish to use computers for proving theorems of mathematics, of formalized physics, or of other fields for which there are completely formalized theories. The ultimate goal of these attempts will naturally be to further our understanding of the fields themselves -- to gather new information about mathematics, physics, or whatever. Since the underlying logic of any of these areas is taken to be classical, one might first wish to investigate the ability of computers to prove theorems in a uninterpreted logical calculus. In this area, however, one might wish to distinguish between attempts to further our understanding of logic *per se* from the attempt to further our understanding of how a logician thinks. In the former case one is unconcerned with the actual method used in arriving at theorems (other than the concern that it be a legitimate method), while in the latter case one would wish to mirror the logician's actual process of constructing a proof. It

seems to me that investigations into pure logic are not very interesting unless it be for the latter type of reason, because there is little new information to be gained about what the theorems of classical logic are.[2'] As Feigenbaum & Feldman (1963: 107) put it:

> The fascination with mechanical theorem proving ... lies less with the end (the production of theorems, perhaps new and important) than with the means (a thorough understanding of the organization of information processing activity in mathematical discovery). It is felt that understanding these problem-solving processes is an important step toward the programming of more complex, more general problem-solving processes for a variety of intellectual tasks.

As I said, it is not always clear what investigators think they are modelling with their logic programs. Newell *et al* (1957) start their article with the apparent claim that they wish to investigate a person's reasoning process, as when they say that their research

> is aimed at understanding the complex processes (heuristics) that are effective in problem-solving ... We wish to understand how a mathematician, for

---

[2']The subject of what theorems can be proved in classical logic was pretty thoroughly canvassed by Whitehead & Russell in 1910-1912, and in modern elementary logic textbooks. While various new and interesting theorems have been discovered in the decades since, the truly exciting work in logic has been done at the model-theoretic level, and not *within* the logic itself.

example, is able to prove a theorem even though he does not know how, or if, he is going to succeed. But if so, it is most unclear what the point is of looking at the algorithmetic methods at all (like their British Museum Algorithm),[3*] since it is manifest that students do not employ such methods. Wang's (1963) approach, on the other hand, would seem to be an investigation into the abstract system of logic, since he professes unconcern for any heuristics or strategies which might be used by students. Instead he wishes to employ methods that guarantee proofs for any decidable subset of logic (p.96). But if so, it is not obvious why he should be so opposed to the Newell *et al* approach -- given that they are interested in a model of a different sort of thing altogether, nor why he should be interested in the "practical feasibility" (for people) of using one set of connectives rather than another (p.97). Investigators employing resolution procedures are schitzophrenic on this issue: on the one hand they are unanimous in their claim that resolution methods are not what people use for constructing proofs; but on the other

---

3*One should point out here that the British Museum Algorithm of enumerating all proofs is not so bad as Newell *et al* (1957) would have us believe. In fact Sikolóssy *et al* (1973) have presented such a program and showed that it could find all the proofs the Logic Theorist could find (and more quickly) plus some others of Whitehead & Russell (1910). The reason for this is that the proofs of the theorems of Chapter II of Whitehead & Russell are so simple as to take only one or two steps. Of course the method bogs down after two-step proofs. No three-step proof was found before memory was exhausted. Perhaps one should examine the proof in Appendix I of the associativity of ↔ to decide whether it is plausible to suppose that any breadth-first approach can work on this.

hand such procedures are advocated for use in trying to simulate human problem-solving systems such as robotic planning and natural language systems.

As an attempt to model a normal (logic) student, the Logic Theorist has the problem of simulating one who is learning some *axiomatic* system. Almost every person (whether professional logician or student) finds axiomatic systems very difficult. If one wants to mirror ordinary logical reasoning, one would do better to look at how people learn to do proofs in one of the other versions of logic. Furthermore if one is interested in how they learn *logic*, one should reject the "semantic" systems also. There are two reasons for this. The first is that the semantic systems are not "really" logic.[2'] I quote here from Georgacarakos & Smith (1978: xiv)

> In keeping with our aim of theoretical soundness, we have sharply distinguished between the semantical and the syntactical correlates of the logical concepts we study throughout the text. We introduce the technique of tree construction as a semantical device, the aim of which is to discover counter-interpretations for invalid argument forms. Many authors regard trees as syntactical devices, and of

---

[2']This claim is over-strong for two reasons: (a) these semantical systems are, in a sense, logic -- as the quoted text makes clear, and (b) it is undeniably true that students do use semantical considerations in formulating "real" logic proofs. (In this last regard see Reiter 1973). However, as I see "real" logic, it is syntactical -- and that presupposes that it is a pattern matching task.

> course in a sense they are (they involve manipulation of symbols). However, the correct purpose of tree construction is semantical in that it is to be used as a device to find possible counterinterpretations.

Recalling Chapter II, we note that resolution systems, although they might not explicitly manipulate trees, are nonetheless a form of semantical system designed to find counterinterpretations, and so this argument works equally well against considering them to be "real" systems of logic. The second reason to reject the resolution systems as an account of a person's logical abilities is that they lend themselves too easily to methods which are beyond the ken of ordinary people. If one really wishes to study how ordinary people perform logical analysis, one should use some natural deduction technique such as the Kalish & Montague method outlined in Chapter II. Even the most optimistic of the resolution theorists, Chang & Lee (1973), are willing to concede that resolution techniques are too complex and time consuming for ordinary people to use.[30]

A reason of a different sort for rejecting the resolution systems is technical: there simply is no good way to construct proofs in these systems. It is to arguments of this sort that we now turn.

---

[30]Anderson's (1973) review of Chang & Lee lists their optimism about the suitability of resolution for mechanical theorem proving as the major shortcoming of the book.

C. Some Problems with the Resolution Strategies

The problems with resolution can be broken into two types: (a) certain strategies are incomplete, (b) other strategies are exponentially difficult.[3'] While I wish here to concentrate on the latter, I should first talk a bit about the former.

The simplest sorts of resolution -- simplest from both the conceptual and computational points of view -- are input and unit resolution. We have already seen that they are not complete, but there is even in these cases something more to be said about implementing them. Ask yourself, is input resolution a *finite* procedure: does it always terminate either in ⊠ or in there being no more resolutions to perform. Well, no -- strictly speaking. Not until we have some way of saying "do not perform a given resolution more than once". This means that track must be kept on which resolutions have already been performed. (Alternatively, if a resolvent is identical to an already-present clause, do not add it. Quit when each member of the input has been checked and found to generate no *new* resolvents.) Quite obviously, this already makes the implementation of even the simple input resolution a task that requires some

---

[3']Of course, unless P=NP, *all* strategies are exponential (in terms of the number of steps required to produce a proof, relative to the size of the premises and conclusion set, since the unsatisfiability problem is co-NP complete even for the sentential calculus, at least in the worst case. I wish to suggest that these systems are exponential in even the average case, and that there is every reason to believe that natural deduction is better. Further discussion appears below.

considerable overhead in terms of checking these things. It is simply not true that "pure resolution" can be implemented in the simple and straightforward manner certain authors seem to claim. Complaints that natural deduction methods in contrast to resolution methods need to keep track of a wide variety of rules of inference, that they do not have simple, uniform logical statements, and that they do not have a uniform proof-completion recognition criterion, are simply not well-taken. (Cf. Sandford 1980: pp. 2-3).

Besides the obviously incomplete strategies such as input and unit resolution, there are less obviously incomplete examples. The "semantic" resolution methods mentioned in Chapter III which use ordered clauses are also not complete (see Chang & Lee 1973: p. 116). And Boyer's (1971) "lock resolution", while itself complete, seems incapable of incorporating any of the standard strategies for simplifying the search space without losing completeness. Thus, for example, one cannot even eliminate tautologies from an unsatisfiable set of clauses (nor use set of support, etc.) and be guaranteed still to generate ☒ by this method. Perhaps fancier strategies for keeping track of "lock numbering" in this method will allow this elimination. But at present this is an open question. (See Sandford 1980: p. 225).

As I see it, the real problem with pure resolution is that it cannot distinguish the conclusion to be proved from the premises -- they are both represented by sets of

clauses. And if there is a contradiction to be found, we have no reason to suspect that it is only because of the addition of the negation of the conclusion rather than amongst the premises. Contrast this with natural deduction where an explicit distinction is made between what is to be proved and what are antecedent lines. To be sure, once we have assumed a negation in natural deduction, such an assumption is treated as any other antecedent line is. But we still know what we are attempting to prove, and resolution provers cannot do this since they do not keep explicit track of the conclusion. A limited attempt to distinguish between premises and conclusion is made in resolution by the addition of the set of support strategy (Wos *et al* 1965). But even here there is no real track kept of what it is we are trying to prove, just that it is in the set (S-T), where T is the set of support and S are all the clauses of the argument. This problem, of not knowing exactly what we are attempting to prove, is most clearly demonstrated by the kind of examples Bledsoe (1971) used to motivate his splitting heuristics. If we are trying to prove (say) a conjunction, it is clearly preferable to notice that we are done when each conjunct separately is shown. Even in a resolution framework it would be easier to prove first that $(S\&\neg P_1)$ is unsatisfiable and then that $(S\&\neg P_2)$ is unsatisfiable rather than that the complex $(S\&(\neg P_1+\neg P_2))$ is unsatisfiable. (This is precisely the move that Bledsoe 1971 makes: after splitting, the subproblems are sent to a

resolution subroutine). And such considerations are even more important when we consider proofs involving complex formulae such as biconditionals which themselves contain many subformulae.

Other ways of organizing the resolution search do not fare well either. Consider linear resolution (whether or not augmented by ordered clause heuristics). While pure resolution ("level saturation resolution") invokes new clauses at a rate exponential in the number of clauses of the previous level, linear resolution appears to be restrictive. Since one of the parents will always be the most recent resolvent, it is clear that once a resolution is started, the number of new clauses at a given level will never be more than the number already present (since that is the upper bound on the number of possible resolutions available) and so the total number of clauses at a given level is just the level plus the number of original clauses. But this gain is only an illusion. The completeness of linear resolution only entails that there is *some* linear resolution which will generate ☒. In fact then, to find it one must try all the possible ways to start a resolution, and at each level all the possible ways to continue it. So the problem is once again seen to be exponential in the number of original clauses. For, not only do we have to consider all the possible ways to start (a polynomial problem) but also (for each started resolution chain) we have to consider every variant of each step. That is, for

each generated resolvent in each chain, if more than one clause will resolve with it, we must keep track of all the possible chains.

Similar remarks can be made about the semantic strategies. Of course every setting will divide the clauses into two non-empty classes: those true in the setting and those false in the setting. Obviously, unless the setting is chosen with care, there is no gain. And the problem of choosing a setting is itself an exponentially explosive one, for if it were easier then the original problem with resolution would not arise. (The number of settings is exponential in the number of distinct literals, so finding the optimal setting is exactly equivalent to showing the original set of clauses to be inconsistent.)

It seems to me that there is no easy way around these difficulties for resolution. Now, it may be that natural deduction techniques introduce an exponentially explosive problem also -- indeed, one would expect so if the simple problem of joint propositional satisfiability is an NP-complete problem. But natural deduction parcels the difficulties into different areas: some problems become recognizing when a subproof is done, others become the generation of subgoals, and still others become akin to resolution-based system's "blindly try all the rules of inference".[32]

---

[32]Of course, not all resolution-based systems can be obviously treated as "blind" in this sense. For example, the "semantic setting" strategies (including set of support) are, in a sense, goal-driven. But this appearance is shown

Now, in the worst cases one expects natural deduction to perform no better than resolution. But natural deduction has a variety of distinct parts and strategies for proving things, rather than just the "keep resolving until ⊠ is generated." Each of these components has an area in which it excels, and one therefore expects that, in the "average case", natural deduction will perform better than resolution.

In any case it seems worthwhile to try, and the system to be described later is a first attempt to show the power of natural deduction and the simplicity with which a variety of strategies can be incorporated into such a non-resolution system.

D. Resolution in General

The resolution method, when not augmented by any heuristic search control, is very inefficient, time consuming, and storage consuming. I quote here from Kowalski (1978: 163)

> The search space determined by unrestricted application of the resolution rule is highly redundant. Redundancy can be avoided, at the cost of flexibility, by restricting resolution to top-down

---

[32](cont'd)to be illusory for the reason mentioned above: the discovery of the correct "setting" is itself an exponentially difficult problem. Other apparently goal-driven systems face the same difficulty because resolution-based strategies are forced to lump all the different aspects of theorem proving into just the one rule of inference: resolution.

or bottom-up inference.

The general, widespread dissatisfaction with resolution as a general purpose inference method can be clearly seen by looking at any volume containing papers where resolution is being discussed from anything but a "technical, theorem proving via resolution" framework. For example, a random sampling from the articles of IJCAI 5 (1977) yielded the following:

> [Our project] consists of writing a computer program which can solve a wide variety of simple mechanics problems stated in English... Our methodology is to find general, justifiable, inference rules which can be combined with mechanics problems. As is well known, when rules like these are run on a general inference machine [a resolution prover] the result is often a combinatorial explosion. Rules are combined in unexpected ways and the search for a solution is developed along unreasonable paths. (Bundy 1977: 496)

> A considerable amount of recent work in theorem proving has been concerned with methods for increasing the power of inferences which can be made in special cases. This appears to be in response to the widespread recognition that general purpose theorem provers, particularly those using resolution as their inference rule, have extreme difficulty with particular aspects of proofs for which there

are fairly effective algorithms and heuristics.
(Harrison 1977: 529)

[There are] two serious disadvantages, in the authors' opinion, of resolution. First, resolution is a deduction-oriented rule, and there are generally vastly more inferences that can be deduced from a set of clauses than are used in the refutations produced, even when very restrictive strategies are used; we point to the low penetrance factors cited in the literature. Second, while the separation of variables insures that the general resolvent will subsume families of resolvents of instances, such generality puts the inference in a strictly local context -- aside from possible future resolutions, there is no connection with any other deduction performed in the search so far.
(Henschen & Evangelist 1977: 541)

The interest in automatic theorem proving, which was very high in the AI community in the late sixties, has decreased. One of the reasons was the impossibility by now of obtaining a theorem prover of wide applicability. In particular "complete" search strategies based on resolution have been heavily criticized. In fact it is felt that the crucial problem is not to have an efficient prover but to be able to communicate with it and to drive it. It is also felt that efficiency improvements

cannot increase significantly the performance.
(Martelli & Montanari 1977: 543)

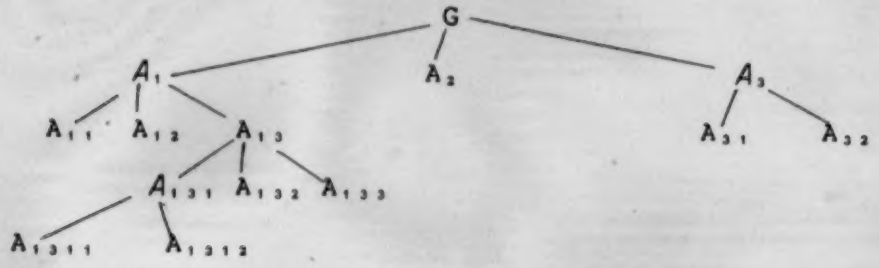
E. Problem Reduction Format

What such researchers as those mentioned in the last section are interested in are the kinds of uses of theorem proving cited in Chapter I that have to do with planning, problem solving, natural language inference, and knowledge representation. And the claim is that resolution systems are not well suited to these uses. Perhaps we should try to see what kind of system *would* be well suited.

In these areas, the overall scheme of knowledge representation and natural language inference can be seen like this: first, there is a representation of "the current state of the world" (a "current-world-data-base", CWDB); second, some new datum is added to the CWDB; and third, the theorem prover is called to find out what other data must be added to the CWDB (or deleted) to accommodate the new datum. In problem solving, there might be a request to try to find a sequence of steps (of alterations to the CWDB) which will have the consequence of adding the desired goal state of the problem to the CWDB. It is clear here the ways in which one might want to invoke a theorem prover; one might want first to check whether the desired goal state isn't already implied by the CWDB, or one might have incorporated into the CWDB such propositions as "if x is at place p_1 at time t_1 and y is at p_1 at t_1, then x is next to y" and "if x is next

to y at t_1 and y is moved at t_2 and $t_1<t_2$, then x is not next to y at t_2" and might have proposed an activity of moving y. The theorem prover should discover which particular statements about the relative locations of x and y to add and delete from the CWDB.

What I take tó be the crucial feature of these approaches is their "problem reduction format." The typical development of problem reduction is to regard this format as being equivalent to an AND/OR tree, where the root node is the goal to be finally achieved. Nodes on a tree are either AND nodes -- which means that every one of the subgoals under this node X (i.e., the nodes X dominates) must be achieved in order for X to be achieved, or OR nodes -- which means that achieving one of the nodes under X is sufficient for achieving X. The leaf nodes of an AND/OR tree represent "primitive actions" (primitive at least from the point of view of the tree as thus far expanded). So, an AND/OR tree for a certain goal G might be (representing AND nodes by italics and OR nodes in roman type):



This representation corresponds to having premisses in an argument of:

$A_1 \rightarrow G$

$A_2 \rightarrow G$

$A_3 \rightarrow G$

$(A_{1,1} \& A_{1,2} \& A_{1,3}) \rightarrow A_1$

$(A_{3,1} \& A_{3,2}) \rightarrow A_3$

$A_{1,3,1} \rightarrow A_{1,3}$

$A_{1,3,2} \rightarrow A_{1,3}$

$A_{1,3,3} \rightarrow A_{1,3}$

$(A_{1,3,1} \& A_{1,3,2}) \rightarrow A_{1,3}$

One tries to find, in the CWDB, some further ("atomic") assertions (such as $A_{1,3,3}$, $A_{1,1}$, $A_{1,2}$) which -- when added to the above premises -- will allow the deduction of G. Or alternatively, the theorem prover might attempt to construct a proof of G from these premises in order to discover what further statements would have to be added in order for the proof to go through and then recommend that these further statements be added. (In this last I have in mind systems like Robbie Robot's of Chapter I, where upon noticing that the goal cannot be achieved unless the CWDB contains some further assertions the system performs the relevant actions so as to alter the CWDB in the requisite manner).

I think the problem reduction format is extremely natural as a representation of how people actually try to solve problems (although as we shall see in a future section, this precise version of it suffers from some technical difficulties). I would argue here that natural deduction systems of logic are more clearly suited to this method of problem solving, natural language inference, and

knowledge representation (as sketchily described here) than are either axiomatic or "semantic" (including resolution) logics. The overall view I take is that (1) such problem solving systems cry out for a *general, domain independent* supervisory system of the natural deduction type which can be supplemented with a domain specific "expert" set of premises (or rules of inference) relevant to that domain, (2) "pure expert" systems are inappropriate on the grounds that they do not reasonably mirror what is common to all the problem domains, and (3) resolution systems (in particular) are the wrong general supervisory system because of their inherent inefficiency.

It seems clear that if one accepts anything like the problem reduction format as being a general representation of human problem solving and as being a reasonable way to model this by a computer, one is going to demand some general method by which the system can figure out (a) how to proceed in attaining the desired goal, and (b) when the goal has been achieved. Given the problem reduction format, this amounts to precisely the natural deduction of the Kalish & Montague sort, where the goal is explicitly stated and the CWDB contains premises for the desired proof. An implementation of the Kalish & Montague system will itself set up appropriate subgoals (given an overall goal) and will construct the proper sequence of steps to achieve each subgoal. Of course if the CWDB is very large, one will want some way of paring down the number of premises actually in

use. The method discussed in Chapter V is my answer to this problem.

Clearly, "pure expert" systems are inappropriate because they do not give an overall picture of the human problem solving ability. Such systems move the pure logical ability (inference) into the programming language itself, and are bound to engender confusions about what logic -- a method of reasoning -- really is. Hayes (1977) argues convincingly that (p. 563)

> The interactions sanctioned by logic between assertions are far richer and more complicated than the interactions between procedures in a procedural language (*any* procedural language). Thus, explicit recursive procedure calls (LISP) are more restricted than explicit coroutine calls (SIMULA), these more restricted than pattern-directed coroutineing (CONNIVER), these more restricted than resolution (which allows both caller and callee to have variables bound during the matching process) and finally resolution itself is a special case of general logic inference rules of instantiation and cut.

Although his following remark does not seem to be necessarily true unless one has in mind *no* strategies for restricting the search space.

> In each case, the more general interaction pattern allows more interactions and hence yields a more

complex search space, and a more difficult search problem.

F. Getting into Clause Form

Very few arguments, whether from a technical area such as mathematics and programming or from "everyday life" such as planning and natural language, are stated in clause form. To use resolution methods then, one either must develop new "non-clausal" methods or else convert one's representations into clause form. In Chapter III I mentioned Murray's (1982) NC-Resolution which operates on non-clausal formulae. This system is extraordinarily complicated but complete, and can be augmented by semantic resolution methods like those discussed in Chapter III. However, it is not obvious that any of the other strategies, e.g., eliminating tautologies, works well on this system due to the difficulty in determining tautologousness in a non-clausal format. It seems to me that the complexity of the resulting method is such that there is very little reason to prefer it to ordinary resolution. After all, the alleged reason to prefer non-clausal format is that most problems are not given in clausal form (by people); so can there be any rationale for a method completely alien to people's proof methods but which clings to the non-clausal form preferred by people? In any case, Murray's system does not really use formulae in the form given by mathematics or natural language. The formulae are converted to a "skolemized form" but without

first converting to prenex normal form. This is the method described by Wang (1963) and also used in the Bledsoe systems discussed in Chapter III. It involves determining "negative" and "positive" occurrences of the quantifiers and replacing the variables bound by some of the quantifiers by appropriate skolem functions of the other quantifiers. It would seem that Murray's system combines the worst of all worlds: formulae must be first converted into a different form from that encountered normally as opposed to the system described later in this thesis, the proof method is extremely complicated as opposed to the resolution method, and few of the well-understood heuristics are applicable to it as opposed to both resolution and natural deduction.

One should point out that some amount of time is required to convert formulae into another form, whether this other form be clausal or the non-clausal "skolemization" of Wang, Murray and Bledsoe. For sufficiently complex formulae this can be a non-trivial amount of time, although it may be trivial compared to the time involved in actually proving the theorem. People who discuss the efficiency of their theorem prover, expecially those who give actual times (as for instance when they compare their prover's times with the well-known times of the Logic Theorist), ought to be required to include the cost of converting to their normalized form. This is especially relevant when comparing resolution provers to those like the Logic Theorist or the system to be described later which do not use any normalized

form.

G. Bledsoe's "natural" systems: A critique

As indicated in the last chapter, Bledsoe's systems have undergone a variety of changes. The earlier systems were not much removed from resolution systems (with the exception of the splitting strategies). The later systems are more in the spirit of true natural deduction. Nonetheless, at least from the published accounts, these later systems are not very good knights to be carrying the natural deduction banner.

I have already noted that Bledsoe *et al* (1972) had mentioned that his system is incomplete. A further discussion of this issue is carried on below in Chapter VII, where it is shown that the system to be displayed later in this thesis is not incomplete, at least not in this way.

Two other shortcomings of the Bledsoe systems should be brought up at this point. First, the earlier systems (1971, 1972) used only backward chaining to generate subgoals (other than those subgoals generated by the splitting heuristics). As is noted in his (1974), this is very inefficient at finding appropriate subgoals; and indeed, seems only to be of real use when a negation has been assumed (that is, when (**imply** (H→¬C) R) was changed to (**imply** (H&C→nil) R). But in this case surely it would be just as easy to switch to a resolution proof. The 1974 system, incorporating "ground forward chaining", presumably

does a better job at this (but because of lack of comparative data, one cannot be sure).

The earlier versions used a depth-first style of search, at least in the "reduction" portion of the systems. As the quotation in Chapter III has it: "if one can make a hypothesis contribute all it can towards establishing a conclusion, then it can be ignored in trying to prove the remainder." This is not a good principle, at least not without some modification. Consider for example linear resolution. We have already seen that in general one needs to re-use centre clauses, even though they "have contributed all they can" towards generating ☒. In natural deduction systems the same phenomenon occurs. One may need to use a formula to generate some intermediate conclusion, and then later use this intermediate conclusion to further the proof using other hypotheses, and finally come back to the original hypothesis again. Such cases are quite common in fact. Suppose for example one wished to prove C: Every person's mother's mother is female, from the premises P_1: Every person has a mother, P_2: If someone is a mother of anyone, then that someone is a female person. It seems that the natural way to prove this would be: let x be a person, then by P_1 there is a y which is a mother of x. Hence by P_2, y is a female person, and hence a person. But now we go back to P_1 to note that y has a mother z, and by P_2 z is a female person and hence female. It is just not true here that what we have done is "used P_1 as much as we can and then

discarded it." (Similar remarks hold here for P_2 as well).

It is perhaps for reasons such as this that in 1974 Bledsoe & Bruell abandoned depth first search in favour of a breadth first search. But such a strategy also is not good. After all, one of the points of natural deduction is that "once you're on the right track, keep it up until you're done". And the reason this can be implemented in a natural deduction system of the Kalish & Montague sort is that one can tell when "you're on the right track", because of the structure of the current subgoal. The reason Bledsoe's systems cannot adopt this is because their whole aim was to return substitution instances for variables from subproofs. But once one rids oneself of this preconception of what is the point of a natural deduction system, one can have the limited depth first search so obviously called for.

H. Natural Deduction and the Problem Reduction Format

One problem with natural deduction, it might be alleged, is that natural deduction systems incorporate too many rules of inference, that they are not elegant, and that one should favour resolution systems with their one rule of inference (augmented by a substitution mechanism) -- even if this tends to lead to less "natural" proofs and a great expansion of the search space. In response to this, one should point out that taste (in elegance or simplicity) is a matter of taste. If one wishes to mirror human problem solvers, the number of rules is not a reasonable measure of

success. Furthermore, if the measure is of programming ease, one should first construct a natural deduction system prover to see whether or not it is easier to construct a resolution or a natural deduction prover. In fact, the prover to be described later in this thesis is quite a bit more complicated to program than the example resolution provers I have seen. But then it can do a lot more than they can. What has not been shown is that given provers of equal capacity, resolution is easier to program than natural deduction. And a further factor to consider here is the ease with which a specific style of theorem prover can be extended when one discovers that it is incapable of proving some class of problems. In this regard, the following quotation from Nevins (1974) is relevant.

> A point worthy of stress is that a deductive system is not "simpler" merely because it employs fewer rules of inference. A more meaningful measure of simplicity is the ease with which heuristic considerations can be absorbed into the system.

Another alleged shortcoming with natural deduction theorem proving is the consequence of these three beliefs which seem to be widely held: first that natural deduction is equivalent to the problem reduction format, second that the problem reduction format is equivalent to AND/OR tree representation of problems, and third that AND/OR trees are not complete in the sense of allowing the solution to every true problem-solving sequence (see Loveland 1978, Chapt. 6).

Of course, if these three beliefs are true, then something is drastically wrong with natural deduction. But it is false that they are all true. The terms 'natural deduction', 'problem reduction format', and 'AND/OR tree' are technical terms meaning different things to different theorists. I have already stated what I take 'natural deduction' to include, viz., Kalish & Montague's system. And I have already stated what I take 'AND/OR tree' to mean. Given these definitions, what are we to say about the above three beliefs? Two things, I think. First, it must be that the term 'problem reduction format' is being used equivocally. The sense in which it is the same as natural deduction is just not the same sense as that in which it is the same as AND/OR trees (as defined earlier). A second thing that might be said is that the explanation of AND/OR trees given is too restrictive, and that a correct explanation will allow them to completely describe the search space.

Let us look at these alternatives. There is a well-established sense in which the problem reduction format just *is* the AND/OR trees as defined. To reduce a problem means to break it into simpler problems such that the solving of all the simpler problems is equivalent to solving the original (AND node) or the solving of one of the simpler problems entails the solving of the original (OR node). In this sense of 'problem reduction format', natural deduction of the sort here described properly includes it but is not the same as it. For, only the SPLITTING heuristics have this

property. In particular, the "reductio" method of boxing and and cancelling cannot be put in this framework. In this sense of 'problem reduction format', natural deduction is not equivalent to problem reduction. But there is another sense of 'problem reduction format' in which any method that includes the use of AND/OR trees is a problem reduction format method. In this sense natural deduction *is* equivalent to problem reduction. But obviously then the second belief is wrong: problem reduction is not then equivalent to AND/OR tree representation.

Perhaps, though, one should expand what an AND/OR tree is. The real problem with allowing AND/OR trees to represent problems is that one only allows nodes to represent "simple states", and one represents the kinds of relationships between these "simple states" by the kind of node (AND or OR). But this means that the only types of assertion we allow will be either $((A_1 \& A_2 \& \ldots \& An) \rightarrow B)$ or $(A_1 \rightarrow B) \& (A_2 \rightarrow B) \& \ldots \& (An \rightarrow B)$. And the only rule of inference recognized is MP in the following form: if one can solve all (AND node) or some (OR node) subproblem, then one has solved the problem. Clearly not every logically valid formula can be represented in this way (there are no negations) nor can every valid argument be solved this way.

Indeed, the AND/OR representation is equivalent to Horn sets. And we have seen above that only some of the arguments we wish to represent can be put in this way. Loveland's (1978: Chapt. 6) solution to this is to expand the

definition of AND/OR tree so as to allow negated nodes of an AND/OR tree and to claim that a branch of the tree is solved if it contains both a simple node and its negation. This in effect allows all formulae (since they can all be defined in terms of →, &, and ¬ of atomics), and admits all proofs (by the addition of the "reductio" proof method). An alternative approach would be to allow arbitrarily complex nodes (any truth functions or quantificational formula) and note that branches like

C
|
(A&¬A)

are automatically valid. (One needn't use indirect proof here, MP and conditional proof will suffice). Or one might incorporate both devices into a larger system, as is done in the theorem prover that will be explained in the next few chapters. What this shows is that it was not natural deduction which was suspect, but rather that the traditional AND/OR tree representation is not an appropriate way to describe the problem reduction format that natural deduction employs.

V. THE GUTS: DATA STRUCTURES AND LOW LEVEL ROUTINES

A. Introduction

In this chapter I discuss the underlying data structures and low-level subroutines used by the theorem-proving system THINKER.[33] The next chapter will be devoted to a discussion of the high-level heuristics that do the actual driving of THINKER. There is one part of the data structures whose discussion will be postponed until Chapter VII, when I talk about difficulties that THINKER, as described here, has in proving certain "tricky theorems".

The source language of THINKER is the SPITBOL dialect of SNOBOL4. This is in keeping with my general attitude (expressed in Chapter IV) that "real logic" is a matter of pattern matching amongst strings, as opposed to an attempt to try to "understand" the problem posed and make use of that "knowledge" to infer the truth of some other item. SNOBOL is a string manipulation language which allows one to define "patterns" as they occur in strings. As we shall see, it is these patterns that we attend to in constructing a proof. SNOBOL also has a built-in data type of TABLE,[34] which makes use of a hash function to index storage

---

[33] These lower level routines and data structures were designed and developed by Dan Wilson. We have discussed, on numerous occasions, the issues involved with THINKER and the exact form of these underlying structures. A paper we jointly wrote on the topic was presented to an inter-disciplinary conference in January 1982 in Fiji, and is to be published as Pelletier & Wilson (1982).

[34] Tables may be thought of as content addressable memories. See Griswold *et al* (1968) for further definition of SNOBOL4.

locations by means of strings. THINKER often asks the question "Does formula 'X' occur thus far in the proof?", and so one wants an efficient way to find this out. If 'X' is stored as a string, then one can use a TABLE to look up where 'X' has occurred. Furthermore, by means of the DATA construct, one can build data types of arbitrary complexity; and as we shall see, this is used quite extensively in THINKER. Control in SNOBOL is rather primitive; it is sequential with branching allowed on the Success or Failure of a pattern match. Of course, since patterns can be quite complex (including recursive patterns), many different things can happen in the course of finding out whether a particular pattern has matched or failed to match a given string. SNOBOL allows unrestricted recursion -- a feature the heuristics make heavy use of. As far as I am aware, the only feature of SPITBOL that THINKER uses which is not also a feature of SNOBOL4 is the function LEQ ("lexically equal").

B. Explicit Manipulation of Formulae

As hinted in the last paragraph, THINKER stores its formulae as strings rather than the list structures or tree structures of other theorem provers. Strings are easier to copy and manipulate than are lists and trees. There is a pattern BF which is a recursive definition of well-formed-formula;, it succeeds if the formula entered as a premise or conclusion is well-formed. If not, a correction is

requested. All the other patterns assume that the formula being operated on is well-formed. There are three further patterns USPLIT, QSPLIT, and BSPLIT, which (given a formula) locate the main connective ('¬' for USPLIT, the quantifier for QSPLIT, or the binary connnective for BSPLIT). This is very fast to do in SPITBOL: since we assume the formula to be well formed, it is merely a matter of "balancing" parentheses by the primitive function BAL. The results of this pattern match are: (a) if it fails, the formula was atomic, (b) if it succeeds, then the main connective is stored in the global variable OP, (c) if it is a BSPLIT, the global variables LOP and ROP contain the two subformulae, (d) if it is either USPLIT or QSPLIT, then ROP contains the (unnegated or unquantified) formula, and (e) if it is QSPLIT, then the global variable VAR contains the variable of quantification. The three patterns are disjunctive parts of the pattern SPLIT which applies the three in turn until one succeeds or they all fail. It is in this way that THINKER finds out "what kind" of formula it is currently operating upon.

The next kind of data structure is the *occurrence table*. There are two types: general and specific occurrence tables. These tables record information about the occurrence of terms (variables and constants -- THINKER does not have arbitrary function terms) in the proof being constructed. These are TABLES, and so indexed by the term in question. In a general table, the value stored at each index is a count

of how many times that term has occurred in the proof; in a specific table, the value is a first in, last out stack each entry of which is the "proof level" (i.e., how many embeddings of uncancelled 'show' are there currently) at the time when the term was added to the table. The functions PPC and PFV add and delete from general or specific tables respectively. There are the following occurrence tables.

CLIST -- a general table of all occurrences of currently antecedent constants (including those in premises)

VLIST -- a general table of all occurrences of all variables

FVLIST -- a specific table of all occurrences of currently antecedent free variables

PremVLIST -- a general table of all occurrences of variables in premises

PremFVLIST -- a specific table of all occurrences of free variables in premises

Various patterns are used to get information about a given formula, and many of these patterns have side effects of entering and deleting things into (from) the occurrence tables (by calling PPC and PFV).

CF (pattern) given an input formula and term, produces a list of the positions of unbound instances of the term in the formula.

PF (pattern) matched against a formula makes appropriate entries into CLIST, VLIST and FVLIST as side effects.

DF (pattern) matched against a formula makes appropriate

deletions from CLIST and FVLIST as side effects.

HF (pattern) matched against a formula (premise) makes appropriate entries into PremVLIST, PremFVLIST, and CLIST as side effects.

FF (pattern) matched against a formula produces the general occurrence table FVL for all free variables in the formula.

These patterns use a recursive descent parser while adding the relevant information to the tables.

C. The Proof Matrix

Two global variables are CURLINE, the current line of the proof, and CURLEVEL, the current "show level" (the depth of embedding of uncancelled 'show' lines). The proof matrix PRMAT is an $n \times 6$ two-dimensional array, where n denotes the number of lines in the proof. The first four entries for each line are: the formula at that line, the annotation of that line (i.e., the justification for that line -- numbers of the antecedent lines used and the rule of inference employed in obtaining that line), the CURLEVEL of that line, and a boolean value indicating whether that line is (is not) antecedent. (When a line is first entered into PRMAT then it is antecedent if it is not a 'show' line, but later additions may make it become non-antecedent). The final two entries will be discussed in Chapter VII; they are not required to understand the standard operation of THINKER.

The function ADDPROOF(X,Y) adds formula X with annotation Y to PRMAT, along with the CURLEVEL and the correct boolean value (and computes the remaining two fields). If the formula ADDPROOFed is not a goal, ADDPROOF calls the function ADDANTE (discussed below); if it is a goal, ADDPROOF was called from the function ADDGOAL (discussed below).

The function SHOWN() backs through the proof matrix up to the last 'show', altering (by DELANTE) the boolean field of each formula it encounters (making antecedent lines non-antecedent and making the 'show' formula it encounters become antecedent). It simultaneously makes changes to the specific and general occurrence tables. It then calls ADDANTE on this (now cancelled) goal and pops the GOALSTACK (see below).

Finally, there is the function PRINTPROOF() which prints out the completely formatted proof.

D. Antecedent Lines and Templates

The data type ANTE is a triple: a line number in the proof matrix, the show level of that line, and a pointer variable (to the next ANTE). A template is a schematic representation of a formula, schematic in that some feature has been omitted and replaced by '@'. Thus 'F(x,@)' is a string which represents the class of actual formulae that have 'x' in the 'F' relation to something. '(@→P)' represents the class of formulae that have '→' as main

connective and 'P' as consequent. THINKER uses such templates in the following way: when a formula -- say '((A&B)→C)' -- is added to the proof matrix and hence is antecedent, THINKER also remembers the templates it can form from it, here '((A&B)→@)' and '(@→C)', thereby remembering that '(A&B)' is the antecedent of some conditional statement and that 'C' is the consequent of some conditional statement. THINKER keeps templates for the following:

1. For every formula, it forms all templates generated by replacing exactly one free term (constant or free variable) by '@'. ('$P^2_1(x,y)$' therefore has two templates: '$P^2_1(@,y)$' and '$P^2_1(x,@)$', but no '$P^2_1(@,@)$' template)
2. Where Q is a quantifier, formulae of the form '$(Q\alpha)\phi\alpha$' generate the template '$(Q@)\phi@$'. (The replacement of '@' for 'α' in '$\phi\alpha$' observes scope requirements, of course). In addition to this "quantifier template" a formula like '(Ex)Fxy' would generate the free variable template '(Ex)Fx@', naturally.
3. A formula whose main connective is binary is of the form $(\phi \circ \psi)$ and generates the templates '$(@\circ\psi)$' and '$(\phi\circ@)$'.

No other formulae generate templates. One easily distinguishes templates from actual formulae by the presence of '@'; thus they can be treated on a par and distinguished only when necessary. The data type TEMP has two fields: a *token* field and a link to the next TEMP. The token is what the '@' is replacing in that template.

ANTELINES is a table whose index is a formula or a template and whose value at that index is a stack. Entries in this stack are either ANTEs or TEMPs. Intuitively, if one indexes ANTELINES by a regular formula, the values will be a stack of places and show levels of where that formula occurs in the proof matrix, while if one indexes ANTELINES by a template, one will get a stack of the tokens from which that template was generated.

The function FINDANTE(X) succeeds if there is an antecedent line 'X'; it returns the ANTE. The function FINDTEMP(X) succeeds if there is an antecedent line with the template 'X'; it returns the actual formula. The functions ADDANTE(X) and DELANTE(X) add to and delete from ANTELINES. ADDANTE stores the formula as an ANTELINE, creates all the TEMPs, stores them via ADDTPLATE, creates the free variable templates and stores them, determines the major connective and CSTACKS the formula (see below), and constructs all the occurrence tables (CLIST, VLIST, FVLIST). DELANTE takes out everything ADDANTE has put in, except VLIST never has anything deleted (it was the list of all variables that ever occurred in the proof).

E. Goals

There is a stack called GOALSTACK which contains the "goals", i.e., the formulae to be proved -- the 'show' lines. (A stack is used since THINKER is always trying to prove the "most current" goal). Each element of GOALSTACK

has seven fields: an index into the proof matrix (the number of the line of the proof it is), the "show level", the actual formula, a table of free variable templates for this goal, two pointers for keeping track of the location in the existentially quantified SIMPLE ring (see below), and a pointer to the next element of the stack. The pointers to the SIMPLEs are merely bookkeeping pointers to make sure that we never do an existential instantiation of a particular formula more than once under a given show level.

ADDGOAL(X) stacks X onto the GOALSTACK and ADDPROOFs X. TESTGOAL(X) finds out whether X is already a goal (so as not to add it again as a goal). GETGOAL() retrieves the formula that is the most recent goal. Recall that SHOWN() pops (deletes) the most recent goal and ADDANTEs that formula.

F. SIMPLEs and FINDing

The data type SIMPLE (for simplification) are stored in a doubly-linked circular list. Each simple contains four fields: a formula, the number of the proof line it occurs on, a forward link and a backward link. There is one such ring for each of the connectives, so that (for example) every antecedent formula that has an '→' as its major connective appears on the '→' ring. Formulae are added to one side of the entry point to the ring by means of CSTACK (recall that ADDANTE calls CSTACK) and are deleted from the ring by DELCSTACK, which is called by DELANTE.

These rings are used by a set of functions called FINDX, where 'X' is replaced by the name of some rule of inference. Thus FINDDNS ("double negations") goes around the negation ring; for each formula it encounters it tests whether that formula starts with two negations. If it finds one such, it checks to see if the un-double-negated formula is already antecedent (if so, go to next member of the ring); if it is not, it ADDPROOFs the un-double-negated formula, which in turn CSTACKs this new formula onto the appropriate ring (unless it is atomic), calls ONESTEP on this new formula (see the next chapter) and if ONESTEP fails goes on to the next formula in the ring, until the ring has been exhausted. Of course a quadruply-negated formula will be doubly-unnegated and CSTACKed onto the end of the same ring as a double negation by FINDDNS, and eventually will be completely unnegated. In a similar vein, FINDBCS circles the '↔' ring, finding '(ø↔ψ)'. It checks to see if '(ø→ψ)' is antecedent, if not it ADDPROOFs it, which in turn CSTACKs it onto the '→' ring, and then calls ONESTEP on it. If this fails it does the same to '(ψ→ø)'. FINDCONTRA goes around the '¬' ring and looks to ANTELINES for the unnegated formula. If it finds it, it is "repeated" as the last line of the proof and SHOWN() is called. FINDMPS goes around the '→' ring always looking for whether the arrow's antecedent is in ANTELINES; if so, it checks whether the consequent is in ANTELINES and, if not, ADDPROOFs it and calls ONESTEP. FINDALLEI goes through the existentially quantified ring and

performs an existential instantiation each to a new variable (if this has not already been done as an antecedent line). The results are CSTACKed, but ONESTEP is not called because it cannot succeed. (The new formula has variables distinct from any in the proof, so it cannot immediately yield the latest 'show'). FINDUI goes around the universally quantified ring and instantiates to everything on the FVLIST, the PremFVLIST, and the CLIST. If these lists are empty, another strategy is used (see the next chapter). The other FINDs, e.g., FINDANDS and FINDMTPS work similarly.

G. TESTing and SEARCHing

TESTUI(X) and TESTEG(X) succeed if the current goal could come from X by universal instantiation or existential generalization respectively. Thus, if '(Ex)(Fx&Gx)' is the current goal then TESTEG('(Fa&Ga)') succeeds; but TESTEG('(Fa&Gb)') would fail. Note that if the goal is '(Ex)(Fx&Ga)', then TESTEG('(Fa&Ga)') succeeds.

One of the strategies used by THINKER requires it to be able to locate negations of biconditionals, of conditionals, and of disjunctions. The function SEARCHNEGS goes around the '¬' ring checking whether there are any, and returns that formula unnegated (unless it is already a goal, in which case it continues around the ring). Another of the strategies involves looking for a conditional that does not also have its consequent as an antecedent line. SEARCHARROW goes through the '→' ring looking for this and returns it.

Similarly SEARCHWEDGE looks for a disjunctive formula where neither disjunct has occurred as an antecedent line. (There are further conditions on these functions which will be discussed in the next chapter).

This completes discussion of the low- and intermediate-level functions, patterns and data types used by THINKER. In the next chapter I discuss how these lower-level functions are employed by the heuristics.

VI. THE BRAINS: HEURISTICS

There are two structurally distinct parts to a Kalish & Montague proof: the 'show' lines and a sequence of antecedent lines. As indicated in Chapter V, these are kept separate in THINKER by two different data structures: the GOALSTACK and the ANTELINES table. In a Kalish & Montague proof, one is always working at proving the most recently-added goal, by adding more and more antecedent lines until a certain configuration of these lines is attained. When this happens, that goal is proved -- it becomes an ANTELINE (i.e., gets "cancelled") and all lines which had been added to ANTELINE after the goal was added to GOALSTACK get deleted (i.e., get "boxed"). When the first-to-be-added goal becomes antecedent, the proof is finished. So when an argument to be proved is entered, the premises (if any) are stored in the premise table, and the goal is put on GOALSTACK. In the normal course of events, the premises do not immediately yield a proof of the goal. Whenever THINKER cannot immediately prove a goal (by methods given below), it has a choice based on the main connective of the formula to be proved: it can either make an assumption or else it can set up one or more subsidiary goals. Although GOALSTACK is a stack, THINKER makes sure that a proposed new goal is not identical with one which is already active (by sequentially going through GOALSTACK using TESTGOAL). If the goal is already active, it uses a different strategy (normally, to make an assumption). If

THINKER is allowed to set a new goal, it uses the following strategy:[35] If the main connective is '↔', it sets itself the two subgoals '→' and '←' to be proved independently. When they are proved (and hence antecedent), THINKER uses the rule CB ("Conditionals to Biconditional") to establish the '↔' formula, and this allows the "cancelling" of the original '↔' goal. Similarly, if the goal's main connective is '&', it sets as subgoals each conjunct. After proving them, it uses the rule Adj ("Adjunction") to establish the '&' formula, which allows the "cancellation" of the original '&' goal. If the formula has a universal quantifier as its main connective and the variable of quantification is not free in any antecedent line (THINKER looks to the occurrence tables for this information), then it sets the unquantified formula as a goal. If it proves this, then since the variable (not being free in antecedent lines) was chosen "arbitrarily", the original universally quantified formula goal is "cancelled" and becomes antecedent.

Those are the only splitting heuristics, but there are other ways for THINKER to set up subgoals. But before discussing them, let us look at the other option open to THINKER -- the making of assumptions. If the goal formula has any connective other than the foregoing, or if the above subgoals are already active, or if the formula is atomic, then (subject to the two provisos to come) THINKER will make an assumption. The Kalish & Montague rules for assumption

---

[35] Some of these strategies are identical with Bledsoe's (1971) "splitting heuristics".

making (see Chapter II) allow one always to assume a negation or to assume the antecedent (if the goal is a conditional). THINKER does this as follows (subject to the two provisos): (a) if the goal formula is a conditional, the next line will be the antecedent of that goal (with the annotation 'ASSUME'); (b) if the goal formula is a negation, the next line will be the unnegated formula (with annotation 'ASSUME'); (c) in all other cases the next formula is the negation of the goal (with the annotation 'ASSUME'). All these assumptions are of course added to ANTELINES.

The two provisos to this assumption-making are: except for one special circumstance which is not relevant here, no line that is already antecedent can be entered again in the proof. So if the would-be assumption is already antecedent it will not be made. Secondly, recall that if '(ø→ɸ)' is a goal, it can be cancelled when 'ɸ' occurs -- regardless of whether 'ø' has been assumed. Thus, before such a 'ø' is assumed, a quick check is made of whether 'ɸ' can be derived by SIMPLEPROOF(ɸ), for which see below. If this fails, then 'ø' is assumed.

Certain parts of a proof are easier to complete than others. When the proof has progressed so far that just one more step is required, it is easy to find that missing step, since there are but a small number of possibilities. In a resolution system one would look for two clauses of length one that are complementary. In THINKER there are more possibilities because there is more than one rule. In

general, THINKER knows the goal to be proved and needs then to search for antecedent lines which in one step will yield the goal. For many of the rules of inference (MP, MTP, MT) this requires finding pairs of antecedent lines, and hence is theoretically an n^2 problem. But there are ameliorations to theory. Thus if ø is the goal and we want to know whether it comes by MP, we need only see if ANTELINES contains the template '(Θ→ø)'. Since this is done by hashing it is quite quick. If it succeeds, we look at the token of 'Θ' and hash again to see if it also is in ANTELINES. The efficiency of this then depends on the size of the ANTELINES table in comparison to how full it is; in SPITBOL the size of a table changes dynamically, so that when it gets too full for efficient search, it is expanded. All of the rules of inference can be seen to operate in this manner; so, given that one knows what formula is to be proved, the template device can very quickly check whether it can be done in one step from some rule of inference. This is THINKER's function SIMPLEPROOF(ø), which attempts to find a one-step justification of 'ø'. If it succeeds, 'ø' will be entered into the proof matrix as the latest ANTELINE. Generally, the point of this is that having 'ø' in the proof allows one to cancel the most current goal. In the example mentioned above (under "provisos"), '(ø→ψ)' was the most recent goal and can be cancelled if 'ψ' occurs (unboxed) beneath it. Thus THINKER calls SIMPLEPROOF(ψ) which, if successful, adds ψ and thereby allows the goal to be cancelled.

One special type of SIMPLEPROOF occurs so commonly that it was thought worthwhile to try to speed it up even more by writing it as a separate function. The case is that we wish to prove the most recent goal and we wish to know whether the last line we added to ANTELINES was the information necessary to do it. So here we not only know what is to be proved, but also know one of the formulae to use in doing the inference. If the goal is ø and the last line is ψ, we can cut out some of the hashing required by SIMPLEPROOF by (a) directly seeing if ø can come from ψ by a one-premise rule of inference, or (b) seeing if a particular line (not a template) is in ANTELINES for two-premise rules of inference. Suppose the goal is ø and the last line is ψ. Now see if ø=ψ; if not, see if ¬ψ is an ANTELINE; if not, check the main connective of ψ. If it's a universal quantifier, call TESTUI(ψ) (see Chapter V); if it is a '&', see if one of the conjuncts is ø; if it is a double negation, see if ø is the unnegated formula; if ø is existentially quantified, call TESTEG(ψ) (see Chapter V); if ø has '+' as its main connective, see if ψ is one of the disjuncts. No one-premise rule would require lookup in ANTELINES for this type of function, but rather success can just be checked by patterns on two given strings. The two-premise rules do require some lookup, but not much. For example, if ø is the goal and ψ the last line, and if ψ is of the form (X→ø), we look for X in ANTELINES; if ψ is of the form (X + ø) we look for ¬X in ANTELINES; if ψ is of the form ¬X we look for (X + ø) or

(¬ø→X) in ANTELINES. And so on for all the rules. This special case of SIMPLEPROOF is called ONESTEP(ø), where ø is the line to be used in constructing a one-step proof of the most recent goal. Keep in mind that the argument to SIMPLEPROOF is the *formula to be proved* (which may not be the most recent goal), while the argument to ONESTEP is the *formula to use* in proving the most recent goal.

ONESTEP is called each time a new line is added to the proof by the FIND routines described in the previous chapter, or when an assumption is made, or when a 'show' line is cancelled. The first of these three cases is clear: when a new line is added, see if that new line will prove the most recent goal. The other two cases require comment. If an assumption is made, it is either the assumption of the antecedent of a conditional 'show' line or else the negation of the 'show' line. In either case, it is possible (although unlikely) that the 'show' line itself can be directly obtained from the assumption. In the former case, suppose the 'show' line is '(p→q)' and THINKER assumes 'p'. If '(p→(p→q))' is an antecedent line, then ONESTEP('p') will enter the line '(p→q)' and cancel the 'show'. In the latter case, suppose the 'show' line is 'p' and THINKER assumes '¬p'. If '(¬p→p)' is an antecedent line then ONESTEP('¬p') will enter 'p' and cancel. The introduction of 'show' lines by THINKER is rigidly controlled with an eye towards proving something that will yield an immediately useful line. (See below for exceptions to this.) Thus whenever a 'show' line

is cancelled it behooves THINKER to see if ONESTEP on that 'show' line will yield the next higher goal.

TRYRULES is a "blind" procedure which will attempt to apply existential instantiation, quantifier negation, and the propositional rules of inference to the antecedent lines. Each time a line X is added by TRYRULES, ONESTEP('X') is called, which is one way to terminate TRYRULES. Another way to terminate it is if an antecedent line is added to which heuristic TRYNEGFLA (below) is applicable. It should be noted that the existential quantifier rule (EI) is applied only once under any 'show' level; thus if it has already been applied and its results have not been "boxed away", then it will not be called again, no matter how many further levels of 'show' are introduced. Of course, TRYRULES (and SIMPLEPROOF, for that matter) might introduce new antecedent lines to which TRYRULES reapplies. The mechanism for this was discussed in Chapter V: TRYRULES calls the FIND procedures and hence the connective rings. Universal instantiation is rigidly controlled, the overall idea being that universally quantified lines will be instantiated only to specific terms. In Chapter VII, under the heading "the EI/UI problem", I shall discuss an improvement to the method discussed here; but for now we shall say that a universally quantified antecedent line is instantiated (a) to every constant in the proof or premises (the CLIST table), (b) to every free variable in an antecedent line (the FVLIST table), (c) if the CLIST and FVLIST tables are empty, to the

variable of quantification of every universally quantified 'show' line. And finally, if (a), (b) and (c) are not applicable, then some variable is arbitrarily picked.

TRYNEGFLA is a rather clever strategy which, when more direct approaches fail, will search ANTELINES for an occurrence of the negation of a conditional and add the unnegated conditional to the GOALSTACK. (The strategy is that if this new goal can be proved, then it will contradict the negated conditional and allow the cancelling of the next higher goal. Generally speaking, proving a conditional is easy since one gets to make an assumption of the antecedent.) Finding the relevant negated formula is done by the SEARCHNEGS procedure discussed in Chapter V. There are similar strategies for negated biconditionals and negated disjunctions: if THINKER finds a negated biconditional it tries to prove the unnegated biconditional, if THINKER finds a negated disjunction it tries to prove one of the disjuncts. Of course this strategy only works if one has assumed the negation of some (true) goal. So there is a global variable which keeps track of whether such an indirect proof has been started and only attempts TRYNEGFLA if it has been.

TRYCHAINING is a strategy which is THINKER's implementation of forward chaining.[3]

---

[3] "This is not really either of the usual types of chaining: forward or backward. It is not backward chaining, because in that strategy we know what the present goal is, say ψ, and on that basis set the goal $(\phi \rightarrow \psi)$. But in THINKER's chaining no attention is paid to the current goal. It is closer to forward chaining, although still not identical to it. In

When other strategies fail, THINKER looks for a conditional in ANTELINES for which the consequent is not also in ANTELINES. If it finds one such, it adds the antecedent to the GOALSTACK and recursively calls the whole set of heuristics on it. If it successfully proves it, then it can do a FINDMP which it could not do before, and so it again can TRYRULES. There is a similar strategy involving FINDMT, where THINKER looks for a conditional whose antecedent does not already occur negated as an antecedent line. In this case it will add the negation of the consequent to the GOALSTACK, and if it can prove it then it can again TRYRULES and be guaranteed of at least one new line by FINDMT. Another version of this strategy involves disjunctions and FINDMTP, where THINKER looks to ANTELINES to find a disjunction for which neither disjunct occurs as an antecedent line. If THINKER finds one, then it will add the negation of one of the disjuncts to the GOALSTACK. If it proves this, then TRYRULES will yield a new line by FINDMTP.

When THINKER fails, either because the formula to be proved is not a theorem or else its heuristics are inadequate to prove it, it displays the proof as thus far constructed and requests the user to enter another line

---

[3] (cont'd)classical forward chaining, we have ϕ and try to prove as a new goal $(\phi \rightarrow \psi)$, for some arbitrary ψ which is not necessarily a goal. Thinker on the other hand finds $(\phi \rightarrow \psi)$ in its antecedent lines and sets ϕ as a new goal, again without attempting to determine whether ψ will aid it in proving the current goal. In the same sense that FINDMP and FINDMT are "forward chaining" rules, so is the present strategy. Indeed, this is merely Modus Ponens with an intermediate step of first proving the antecedent.

(either an ANTELINE or a new goal). It adds this to the proof with an appropriate annotation, and once again attempts the proof. With this facility in tandem with the "mathematically oriented" natural deduction system, it would seem that the sort of man-machine interaction that so many researchers have touted is possible (see the discussion in Chapter I concerning Fred Faculty, Jr.). I shall not pursue this aspect any further at this point, but wish to emphasize the promise that this holds.

The overall monitor of all these other heuristics is PROOF. It adds goals, calls the lower-level heuristics as necessary, and recursively calls itself as new goals are added. And when it gets "stuck", it calls the above mentioned HELP routine. A more detailed explanation in pidgin ALGOL is given in Appendix II.

I give here a moderately simple propositional calculus example which illustrate how THINKER works. The example is to show the equivalence between disjunction and the conditional: ((P+Q)↔(¬P→Q)). This is put on the goal stack. Since it is a biconditional, PROOF adds a conditional to the goals: ((P+Q)→(¬P→Q)), and calls itself recursively. First a call to SIMPLEPROOF('((P+Q)→(¬P→Q))') is made, which fails. Next follows a call SIMPLEPROOF('(¬P→Q)'), which also fails. Then, since this is a conditional, it assumes '(P+Q)' and asks whether ONESTEP('(P+Q)') will prove the most recent goal. The answer is no, so it asks whether SIMPLEPROOF('(¬P→Q)'). Again no, so it adds '(¬P→Q)' as a

goal, recursively calls itself, and will eventually (after a SIMPLEPROOF call) assume '¬P'. It now calls ONESTEP('¬P') to see whether '(¬P→Q)' is derivable from the ANTELINES. The answer is no, so it calls SIMPLEPROOF('Q'). This succeeds (from the ANTELINES '(P+Q)' and '¬P' by MTP) and so it adds 'Q' to ANTELINES, and then notices it can cancel the (¬P→Q) goal. So it does (and deletes the '¬P' ANTELINE), thus ending that recursive PROOF call. But this proves ((P+Q)→(¬P→Q)), so it cancels that goal (and deletes the (P+Q) ANTELINE). PROOF then decides that to prove the original goal, it needs to prove ((¬P→Q)→(P+Q)), so this is added to the goals, PROOF is recursively called and assumes (¬P→Q). ONESTEP('(¬P→Q)') fails, as does SIMPLEPROOF('(P+Q)'). So '(P+Q)' is added to the goals, and PROOF called recursively. It assumes '¬(P+Q)'. At this stage the proof looks like

```
1.  Show ((P+Q)↔(¬P→Q))
2.      *Show ((P+Q)→(¬P→Q))
3.      |  (P+Q)              Assume
4.      |  *Show (¬P→Q)
5.      |  |    ¬P           Assume
6.      |  |    Q            3,5MTP
7.      |  Show ((¬P→Q)→(P+Q))
8.      |    (¬P→Q)           Assume
9.      |     Show (P+Q)
10.     |       ¬(P+Q)        Assume
```

No rules of inference apply to our three ANTELINES (#2, 8, 10). PROOF notices that line 10 is the negation of a disjunction and so calls TRYNEGFLA, which adds 'P' as a goal and calls PROOF recursively. '¬P' is assumed, and a MP is performed with the lines 8 and 12, yielding 'Q'.

ONESTEP('Q') is called and succeeds, since by doing ADD on it we generate a contradiction and hence can cancel our most recent goal. Having proved P and thus adding it to ANTELINES, PROOF notes that ONESTEP('P') allows us to prove the most recent goal (by ADD). But this most recent goal, '(P+Q)', was the consequent of the previous goal, and so that goal too is proved. We are now on the topmost recursion, trying to prove line 1, and we have two ANTELINES (#2, 7). We apply the rule CB to give us the final line #17, which allows us to cancel line 1, finish and print out the proof.

```
1.  *Show ((P+Q)↔(¬P→Q))
2.  |  *Show ((P+Q)→(¬P→Q))
3.  |  |  (P+Q)              Assume
4.  |  |  *Show (¬P→Q)
5.  |  |  |  ¬P              Assume
6.  |  |  Q                 3,5MTP
7.  |  *Show ((¬P→Q)→(P+Q))
8.  |  |  (¬P→Q)            Assume
9.  |  |  *Show (P+Q)
10. |  |  |  ¬(P+Q)         Assume
11. |  |  |  *Show P
12. |  |  |  |  ¬P          Assume
13. |  |  |  Q              8,12MP
14. |  |  |  (P+Q)          13,Add
15. |  |  |  ¬(P+Q)         10,R
16. |  |  (P+Q)             11,Add
17. |  ((P+Q)↔(¬P→Q))        2,7CB
```

Further examples of more interesting problems, together with comparisons to other theorem provers, will be given in the next chapter and Appendix I.

VII. SOME COMPARISONS AND DISCUSSION

A. The Logic Theorist and the British Museum Algorithm

The Logical Theorist (LT), in its original formulation (Newell *et al* 1957) managed to prove 38 out of the first 52 theorems of Whitehead & Russell (1910: Chap. 2);[37] the "new, improved" LT of Stefferud (1963, also reported in Newell & Shaw 1972) managed to prove 19 of the 67 theorems in Chapter 2 of Whitehead & Russell, plus three from Chapter 3. On the whole this was a pretty meagre achievement, since the theorems are extremely simple. A selection of some of the Whitehead & Russell theorems are proved in Appendix I. As mentioned in Chapter III, the most difficult one proved by the original LT was $((p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p))$, which was apparently not proved by the new LT. The most difficult of the ones proved by the new LT is $(\neg\neg p \rightarrow p)$.

THINKER proved all these theorems without difficulty; the most difficult was $((p+q) \rightarrow (p+r)) \rightarrow (p+(q \rightarrow r))$, which is presented in Appendix I together with a sampling of ones of interest from the two versions of the LT. "Difficulty" here in this context means "how much CPU time was used" and "how many statements were executed". Such comparisons are somewhat unfair to THINKER because this is the full version of THINKER where numerous of the statements executed and much of the time involved has to do with various checks relevant to quantified statements. Since neither of the LTs

---

[37] The data on LT derives from the report made by Siklóssy, *et al* 1973, as does the information about the "new LT".

were concerned to prove quantified statements, they were not encumbered by such considerations. It is estimated (by comparison with an earlier, less efficient version of THINKER which only did propositional arguments) that the mere addition of code sufficient for quantifier checking makes execution slow down by a factor of 2 or 3, even when there are no quantifiers in the formulae to be considered. Nevertheless THINKER proved these theorems much faster than either of the LTs did. Time comparisons here are difficult because of the different machines and languages used. Both versions of LT were programmed in IPL-V and apparently run on a JONNIAC. THINKER was run on an AMDAHL 470 V/8.

THINKER's ability to prove these theorems is not such a great feat. Wang (1963) gives a program which does the same thing, and this program is so simple that it is used as an example in Griswold *et al* (1968: p.183ff). Furthermore even a breadth-first "exhaustive enumeration" of proofs can do better than LT. Siklóssy *et al* 1973 report on such a program and compare its results to those of the two versions of LT. Everything either of the LTs could prove was proved by their exhaustive search method, plus some others. Details can be found in Appendix I. The most difficult theorem which was proved by this program was $\neg(p\rightarrow q)\rightarrow(q\rightarrow p)$, for which see THINKER's proof in Appendix I. The authors remark that in their opinion, the hardest theorem to prove of the first 52 is $(\neg p\rightarrow q)\rightarrow(\neg q\rightarrow p)$, while the hardest of the 67 in Chapter 2 of Whitehead & Russell is $((p+q)\rightarrow(p+r))\rightarrow(p+(q\rightarrow r))$. As noted

above, THINKER concurs in this assessment.

B. Bledsoe's "Natural" Systems

It is difficult to know the precise limits of Bledsoe's systems, since no list is given of what of arguments or theorems can and cannot be proved. However, as mentioned in Chapter IV, Bledsoe *et al* (1972: p.59) remark that their system, IMPLY, is not complete.

> IMPLY is incomplete in many ways. For example, ...it can prove the skolemized formula
>
> (P0&(Px→Pf(x))→Pf(f(x)))
>
> but it cannot handle the following equivalent formula
>
> (¬P0+(Px+Pf(f(x))&(¬P0+(¬Pf(x)+Pf(f(x))))
>
> because the substitution [0/x] satisfying the first conclusion does not satisfy the second.

Again, it is difficult to know just what they mean here. The induction axiom is just that: an axiom. Ergo, it seems implausible to suppose that their system proved it.[38] What I suspect they mean is that IMPLY cannot prove the two formulations to be equivalent. Yet their equivalence is merely a matter of the propositional logic. In Appendix I two proofs are given, one with the equivalence stated in the

---

[38] Difficult but not impossible. For, they may have other methods in the (unstated!) background which yield each instance of the induction scheme. See, for example, Goguen (1980) for details. Even so, the fact remains that IMPLY could not prove a truth-functionally equivalent version of it.

propositional logic as

$$(((p\&(q\rightarrow r))\rightarrow s)\leftrightarrow((\neg p+(q+s))\&(\neg p+(\neg r+s))))$$

thereby showing that the formulations are equivalent in that sense, and the other proof stated in the quantifier logic as

$$((Ax)((Pa\&(Px\rightarrow(Ey)(Py\&Rxy)))\rightarrow(Ez)(Ew)(Pz\&Rxw\&Rwz))\leftrightarrow$$
$$(Ax)((\neg Pa+(Px+(Ez)(Ew)(Pz\&Rxw\&Rwz)))\ \&$$
$$(\neg Pa+(\neg(Ey)(Py\&Rxy)+(Ez)(Ew)(Pz\&Rxw\&Rwz))))$$

To get this latter version I restored the implicit universal quantifiers to the front of their formulae, and treated the "successor function" applied to (universally quantified) *x* merely as an existentially quantified variable in the scope of that universal quantifier. (This was done since THINKER does not have arbitrary functions). Of course, then "f(x)" no longer means the intended "the number after *x*" but only "something dependent on *x*". And "f(f(x))" will not be given the intended interpretation of "the number after the number after *x*", but also as "something related to something dependent on *x*". Although these versions of the two formulations of the induction axiom do not quite say what the original formulations said, the difference between what I give and the original is the same for the two formulations so that the resulting formulae are still equivalent. The second proof in Appendix I shows THINKER proving this. And just in case this formulation does not ring true, a version of these formulations is proved wherein the functions are replaced with constants.

As mentioned in Chapter IV, it seems likely that IMPLY is severely limited in what it can do. Indeed, I suspect it cannot prove many of the quantificational theorems given in Appendix I, and I am certain that it has not solved what I later in this Chapter call the UI/EI problem. And all this is assuming that it has a "built in" propositional checker to fall back on. (See the discussion in Chapter IV).

C. The Full Propositional Logic

THINKER has proved every propositional theorem (and argument) it has been asked to, and it has been asked every one from Kalish & Montague (1964: Chapt. 2), Whitehead & Russell (1910: Chapt. 2, 3 and 4), and Thomason (1972). There are 115 such theorems in Kalish & Montague, plus 45 exercises; there are 10 Whitehead & Russell theorems not included in the Kalish & Montague ones; and there are 5 Thomason theorems not so included, plus 30 exercises. Appendix I contains proofs of the more interesting ones, together with discussion.

Conversion to clause form requires the truth of certain propositional equivalences such as

$(\neg(p\rightarrow q)\leftrightarrow(p\&\neg q))$

$((p\rightarrow q)\leftrightarrow(\neg p+q))$

$(\neg(p\leftrightarrow q)\leftrightarrow((p\&\neg q)+(\neg p\&q)))$

$((p\leftrightarrow q)\leftrightarrow((p\&q)+(\neg p\&\neg q)))$

$(\neg(p\&q)\leftrightarrow(\neg p+\neg q))$

$(\neg(p+q)\leftrightarrow(\neg p\&\neg q))$

((p+(q&r))↔((p+q)&(p+r)))

These cannot be proved *within* a resolution based system, since they are presupposed by such systems in converting to clause form. That they are not trivial to prove is shown by their proofs in Appendix I. The longest of the propositional theorems to prove is the associativity of ↔, Kalish & Montague's Theorem 95.

(([p↔q]↔r)↔(p↔[q↔r]))

D. The Predicate Logic

As already illustrated by the proofs in Appendix I of the quantifier theorems mentioned in discussing Bledsoe, THINKER is quite good at proving a wide class of such problems. Appendix I also contains a selection of problems gathered from Kalish & Montague Chapts. 3 and 4, and Thomason Chapts. 9-11. THINKER's proofs of some of these are perhaps noteworthy.

The following is a problem from Kalish & Montague which I regularly give my elementary logic students as an extra credit, "hard" problem at the middle of a year long course:

(Ex)(Fx→P), (Ex)(P→Fx) ⊢ (Ex)(P↔Fx)

The simplicity of this problem is deceptive. The premises each say that some object has the property indicated by the open formula following the quantifiers. But there is no guarantee that it is the *same* object, so one cannot infer the conclusion via the conditionals-to-biconditional inference rule. However, since the subformula P does not

have x free in it, we are in this case allowed to make the inference. To see this, I usually tell my students to use this informal argument. If we assume the negation of the conclusion, we will have (by QN)

1. $(Ax)\neg(P\leftrightarrow Fx)$

Now, do a separation of cases: suppose first that P is false, then suppose that P is true. Either case is contradictory, so the assumption is false, hence the conclusion is true. If we suppose P false, do an EI on the first premise (to z_1) and a MT with $\neg P$. This yields $\neg Fz_1$. $\neg P$ and $\neg Fz$, together entail $(P\leftrightarrow Fz_1)$. But a UI on our assumption (1) yields $\neg(P\leftrightarrow Fz_1)$, contradictorily. If we suppose P true, do an EI (to z_1) on the second premise and an MP, yielding Fz_1. P and Fz_1 together entail $(P\leftrightarrow Fz_1)$, which is contradicted by UIing the assumption.

THINKER proved this problem differently. After introducing the assumption (1) above, and the premises, and EIing the premises to new variables, it UIs (1) to each of the new variables giving us

2. $\neg(P\leftrightarrow Fz_1)$

3. $\neg(P\leftrightarrow Fz_1)$

It then does a SEARCHNEGS on (2), setting the goal

4. show $(P\leftrightarrow Fz_1)$

Since we already have $P\rightarrow Fz$, from the EI of the premise, we need only

5. show $Fz_1\rightarrow P$

6. Fz_1 ASSUME

7. show P

8. ¬P

THINKER now does another SEARCHNEGS, this time on (3) setting the goal

9. show P↔Fz,

Again we already have Fz,→P from the premise, so we only

10. show P→Fz,

11. P ASSUME

THINKER now repeats 8, cancelling 10. Putting this together by CB with Fz,→P we can cancel 9. THINKER now repeats (3), cancelling by contradiction line 7 and hence 5. With 5 and our P→Fz, we CB and cancel 4, which contradicts 2, thus proving the theorem.

THINKER can also prove some rather tedious problems, such as the following from Kalish & Montague.

$$\vdash [([(Ex)Fx \leftrightarrow (Ex)Gx] \& (Ax)(Ay)((Fx \& Gy) \rightarrow (Hx \leftrightarrow Jy))) \rightarrow [(Ax)(Fx \rightarrow Hx) \leftrightarrow (Ax)(Gx \rightarrow Jx)]]$$

$$(Ax)(Fx \rightarrow (Ax)Gx),\ ((Ax)(Gx \rightarrow Hx) \rightarrow (Ex)(Gx \& Ix)),\ ((Ex)Ix \rightarrow (Ax)(Jx \rightarrow Kx)) \vdash (Ax)((Fx \& Jx) \rightarrow Kx)$$

$$[(Ax)(Fx \rightarrow Gx) \vee (Ex)(Fx \& Hx)],\ (Ex)Fx,\ (Ax)(Ix \rightarrow (\neg Jx \vee \neg Hx)),\ (Ax)(Fx \rightarrow (Ix \& Jx)) \vdash (Ex)(Gx \& Fx)$$

$$(Ax)(Fx \rightarrow (Gx \vee Hx)),\ (Ax)((Gx \vee Hx) \rightarrow Ix),\ \neg(Ex)(Ix \& Gx),\ (\neg(Ex)Fx \rightarrow (Ex)Gx) \vdash (Ex(Fx \& Hx)$$

$$(Ex)(Fx \& \neg Gx),\ (Ax)(Fx \rightarrow Hx),\ (Ax)((Jx \& Ix) \rightarrow Fx)\ [(Ex)(Hx \& \neg Gx) \rightarrow (Ax)(Ix \rightarrow \neg Hx)] \vdash (Ax)(Jx \rightarrow \neg Ix)$$

The proofs of these theorems are all in Appendix I.

E. The UI/EI Problem, I

There are some theorems THINKER has great difficulty in proving. Consider an attempt to prove (Ey)(Ax)(Py→Px). THINKER will assume its negation, and apply quantifier negation to it to yield

1. (Ay)¬(Ax)(Py→Px)

which is the only line THINKER will consider available from now on, until it generates new antecedent lines. Having no constants or free variables in the proof, THINKER "picks one" (say, z_1) and does a universal instantiation (UI) yielding the new antecedent line

2. ¬(Ax)(Pz_1→Px)

which allows another QN to

3. (Ex)¬(Pz_1→Px)

THINKER will now perform an existential instantiation (EI) to a new variable, z_2, yielding

4. ¬(Pz_1→Pz_2)

However, there is now a variable in the proof, viz., z_2, to which line 1 ought to be instantiated (so THINKER believes). Hence another antecedent line is generated

5. ¬(Ax)(Pz_2→Px)

In turn, line 5 is QNed and then EIed (to a new variable z_3) yielding

6. ¬(Pz_2→Pz_3)

z_3 is a new variable, hence 1 will be UIed to it. As can be seen, this leads to a fruitless series of UIs alternating with EIs. In this problem, THINKER ought to not continue

this, but rather (once it has 4) set up a new goal

7. Show ($Pz_7 \rightarrow Pz_8$)

using the SEARCHNEGS strategy. In fact, THINKER has at its disposal only 40 variables, so eventually it *will* set itself the goal 7. But this happens only after 39 EI/UI combinations (preceeded by the initial UI) which will make the 40th EI fail. Only then does THINKER move on to the SEARCHNEGS strategy. This is the phenomenon I dub "the UI/EI problem".

It is worth noting that this problem also arises in resolution provers. Suppose such a prover has a "unit preference" strategy: whenever a unit resolution is available, do it; if none are available perform some other resolution. And suppose it has these two clauses available

$\neg Px + Pf(x)$

$Pf(x)$

Such a prover will note that the substitution of f(x) for x in the first clause allows it to perform a unit resolution against the second clause yielding

$Pf(f(x))$

But this allows another unit resolution against the first clause by substituting f(f(x)) for x, and yields

$Pf(f(f(x)))$

As one can see, this too is an unending sequence of unit resolutions. Indeed, this is essentially the same problem used earlier to illustrate the difficulty in THINKER, for the first clause is merely the clausal form of

(Ax)(Ey)(Px→Py)

and the second clause represents being able to get Pz, in THINKER. (The f(x) of the second clause means that its variable was existentially quantified and was originally in the scope of a universal quantifier, as the transition from lines 3 to 4 of the THINKER proof above has it.)

Two methods of avoiding the problem in THINKER suggest themselves. The first would be to have THINKER apply the SEARCHNEGS strategy as soon as it became available, rather than continuing with the UIs. The second would be to keep track of variables introduced in this undesirable manner. That is, whenever a new variable is introduced by an EI and that line has come from a UI, do not allow that universally quantified line to perform another UI to this new variable.

The first method was judged inadequate on the grounds that there is no guarantee that SEARCHNEGS is in general the appropriate strategy. Indeed, perhaps other sentential rules such as MP, DN, etc., should be performed before SEARCHNEGS. In any case, there is no guarantee that SEARCHNEGS will succeed in finding a formula which is the negation of an →, of a ↔, or of a +, so it would be unwise to stop the TRYRULES early.

For these reasons, and in consultation with Dan Wilson, I opted to implement the other strategy. It is obvious, however, that the example cited at the beginning of this section is only one of many possible ways the UI/EI problem might arise. Consider a proof which has these two lines as

antecedent

(Ax)Fx

(Ax)(Fx→(Ey)Gy)

The first might get instantiated to z, which leads to the second being UIed to z, and a MP performed, leaving the proof with

(Ey)Gy

as an antecedent line. An EI is now performed on this line to the new variable z_1. But this now allows the first two lines to be UIed to z_1 and the unending (until all 40 variables are used) series of UI/EI continued.

Generally, what THINKER wants is to keep track of universal formulae *no matter how far back in a chain of justification* that eventually leads to an EI. Having kept track of these, we wish to prohibit such universal formulae from being UIed to the new variable obtained by the EI. Two constructs are necessary for this: the *ancestor list* of a formula (that formula's A-list), and the *prohibited list* of variables for a formula (that formula's P-list). These are the two further fields of the data types ANTE and GOAL mentioned in Chapter V, and promised there to be explained in this Chapter. They are in fact lists; the first is a list of pointers to lines in the proof matrix (the PRMAT) and the latter is a list of variables. The implementation of the A-list is effected by (optional) extra arguments to the function ADDPROOF: whenever a formula is added to the proof, pointers to its ancestor(s) are also computed. Of course,

all one needs keep track of are the universally quantified ancestors. So the relevant method is a matter of copying the pointers (to universally quantified lines) of a line's immediate parents. If line X comes from line Y (in ways soon to be specified) and line Y is universally quantified, then line Y is on line X's A-list. In addition, if line X comes from line Y (in the relevant ways) then every line on Y's A-list is also on X's A-list. In effect then, the A-list of a line in the proof is a list of all the universally quantified formulae which that line depended upon (in the sense of there being an inference chain which invoked the universally quantified line).

Now, whenever an EI is performed on a line, every member of that line's A-list has the new variable added to its P-list. And a restriction on the function FINDUI prevents a UI being performed on a universally quantified line to any variable on that line's P-list.

One computes A-lists as follows:

1. Premises and the initial "show" have a null A-list.
2. If a line X comes by UI from line Y, then line Y and all of Y's A-list are on line X's A-list.
3. If a line X comes by a rule of inference from line Y (and possibly also line Z in the case of a two-premise rule of inference), then line X has all of Y's (and Z's) A-list as its A-list.
4. An assumption has the A-list of the "show" line it was assumed from.

5. If a "show" line is generated by a SPLITTING heuristic, then it has the A-list of the "show" line it came from.
6. If a "show" line is generated by FINDNEGS or CHAINING, then it has the A-list of the line in the proof that gave rise to this "show".[3']

Now, if a line X is EIed to a new variable, z, say, then z, is placed on the P-list of all of X's A-list. And UIs are not allowed on a (universally quantified) line to any member of its P-list.

F. The UI/EI Problem, II

The above emendation allows one to prove the theorem mentioned in the last section, as the proof in Appendix I illustrates. However, consider this problem:

$(Ex)(Ay)(Az)((Py\rightarrow Qz)\rightarrow(Px\rightarrow Qx))$

THINKER will assume its negation and do a QN leaving it the line

1. $(Ax)\neg(Ay)(Az)((Py\rightarrow Qz)\rightarrow(Px\rightarrow Qx))$

THINKER now does a UI (to z_1, say), a QN and an EI (to the new variable z_2)

2. $\neg(Az)((Pz_2\rightarrow Qz)\rightarrow(Pz_1\rightarrow Qz_1))$

z_2 is on 1's P-list (since 1 is on 2's A-list) and so line 1 cannot be UIed to this. THINKER therefore does a QN on 2 and another EI (to z_1) leaving

3. $\neg((Pz_2\rightarrow Qz_1)\rightarrow(Pz_1\rightarrow Qz_1))$

[3']I.e., if "$\neg(p\rightarrow q)$" was an antecedent line and FINDNEGS used it to generate the line "show $(p\rightarrow q)$", then the latter has the A-list of the former. Similarly for the case where "$(p\rightarrow q)$" was used by CHAINING to generate the line "show p".

z_7 is also on 1's P-list (since 1 was on 3's A-list) and so line 1 cannot be UIed to this. THINKER therefore does a FINDNEGS and generates these "show" lines/assumption lines (the first is from FINDNEGS and the rest are the SPLITTING heuristics).

4. show $((Pz_8 \rightarrow Qz_7) \rightarrow (Pz_7 \rightarrow Qz_8))$

5. $Pz_8 \rightarrow Qz_7$ ASSUME

6. show $Pz_7 \rightarrow Qz_7$

7. Pz_7 ASSUME

8. show Qz_7

9. $\neg Qz_7$ ASSUME

And now THINKER is stuck. No rules of inference can be applied to any of these lines. The proof, however, *could* be completed in the following way.

10. $\neg(Ay)(Az)((Py \rightarrow Qz) \rightarrow (Pz_8 \rightarrow Qz_8))$ 1,UI(to z_8)

11. $\neg(Az)((Pz_4 \rightarrow Qz) \rightarrow (Pz_8 \rightarrow Qz_8))$ 10,QN,EI(to z_4)

12. $\neg((Pz_4 \rightarrow Qz_3) \rightarrow (Pz_8 \rightarrow Qz_8))$ 11,QN,EI(to z_3)

13. $\neg(Ay)(Az)((Py \rightarrow Qz) \rightarrow (Pz_7 \rightarrow Qz_7))$ 1,UI(to z_7)

14. $\neg(Az)((Pz_4 \rightarrow Qz) \rightarrow (Pz_7 \rightarrow Qz_7))$ 13,QN,EI(to z_4)

15. $\neg((Pz_4 \rightarrow Qz_3) \rightarrow (Pz_7 \rightarrow Qz_7))$ 14,QN,EI(to z_3)

16. show $((Pz_4 \rightarrow Qz_3) \rightarrow (Pz_8 \rightarrow Qz_8))$ (via FINDNEGS on line 12)

17. $Pz_4 \rightarrow Qz_3$ ASSUME

18. show $Pz_8 \rightarrow Qz_8$

19. Pz_8 ASSUME

20. show Qz_8

21. $\neg Qz_8$ ASSUME

22. Qz_7 5,19MP

23. show ((Pz$_4$→Qz$_3$)→(Pz$_2$→Qz$_2$)) (via FINDNEGS on line 15)

24. Pz$_4$→Qz$_3$ ASSUME

25. show Pz$_2$→Qz$_2$

26. Qz$_2$ 22,R

Line 26 cancels line 25, which cancels 23. If line 15 is repeated now, it contradicts line 23 and allows cancellation of line 20, which cancels line 18 and in turn 16. 16 contradicts 12 and hence cancels 8, cancelling 6 and then 4. In turn, this completes the proof by contradicting 3.

The solution to this problem requires being able to do *controlled* UIs to variables on a formula's P-list. They must be controlled or else one into runs the problem indicated in the last section. But they must be allowed. In the above example, line 1 was UIed to the variables z$_4$ and z$_2$, which were on its P-list. The solution I adopted was: retain the restriction given in the last section with regard to the function FINDUI, which was used in the "blind" procedure TRYRULES, but after all the strategies have been tried, see whether the new function FINDUIPROHIB (which does a UI to all variables on a formula's P-list) will add any new lines to the proof. If not, continue on to the HELP routine; but if it does add new lines, branch back and redo the various strategies. Of course, any new EIs will add new variables to the P-lists, but the original restriction is still in effect: FINDUI will not allow UI to such variables. In the above example, after we allowed a FINDUIPROHIB on line 1 to the variables z$_4$ and z$_2$ yielding lines 10 and 13, THINKER

found itself able to do QNs and EIs on these lines. The EIs generated new variables z_1, z_2, z_4, and z_3, which were placed then on line 1's P-list (since lines 10 and 13 both had line 1 on their A-list). Line 1 cannot be UIed to these variables until all the strategies have applied, and in the present case the strategies yield a proof. Had they not, THINKER would eventually have reapplied FINDUIPROHIB and discovered that new lines could be generated. And this would then restart all the heuristics.

THINKER's proof of this theorem can be found in Appendix I.

G. The UI/EI Problem, III

Consider this theorem

(Ax)(Ay)(Ez)(Aw)((Fx&Gy)→(Hz&Iw)) →

((Ex)(Ey)(Fx&Gy)→((Ez)Hz&(Aw)Iw))

THINKER assumes

1. (Ax)(Ay)(Ez)(Aw)((Fx&Gy)→(Hz&Iw))

and generates the goal-assumption-goal sequence

2. show (Ex)(Ey)(Fx&Gy)→((Ez)Hz&(Aw)Iw)
3. (Ex)(Ey)(Fx&Gy) ASSUME
4. show (Ez)Hz&(Aw)Iw

A splitting heuristic now generates the goal-assumption sequence

5. show (Ez)Hz
6. ¬(Ez)Hz

Line 3 is EIed and its conclusion is EIed again (the two EIs

to distinct new variables)

7. Fz,&Gz,

Line 6 is QNed and line 7 is Sed (Simplified)

8. (Az)¬Hz

9. Fz,

10. Gz,

Line 1 is UIed to these two variables, yielding

11. (Ay)(Ez)(Aw)((Fz,&Gy)→(Hz&Iw)).

12. (Ay)(Ez)(Aw)((Fz,&Gy)→(Hz&Iw))

The variables z_1 and z_2 are on 1's P-list, so 1 will not be UIed to them (until after all heuristics have had their chance to apply and FINDUIPROHIB is called). Lines 11 and 12, however, can be UIed to these variables, yielding

13. (Ez)(Aw)((Fz,&Gz,)→(Hz&Iw)) 11,UI

14. (Ez)(Aw)((Fz,&Gz$_2$)→(Hz&Iw)) 11,UI

15. (Ez)(Aw)((Fz$_2$&Gz,)→(Hz&Iw)) 12,UI

16. (Ez)(Aw)((Fz$_2$&Gz$_2$)→(Hz&Iw)) 12,UI

The lines 13-16 now allow EIs (to new variables)

17. (Aw)((Fz,&Gz,)→(Hz$_3$&Iw)) 13,EI

18. (Aw)((Fz,&Gz$_4$)→(Hz$_4$&Iw)) 14,EI

19. (Aw)((Fz$_5$&Gz,)→(Hz$_5$&Iw)) 15,EI

20. (Aw)((Fz$_6$&Gz$_6$)→(Hz$_6$&Iw)) 16,EI

Lines 17 and 18 have lines 11 and 1 on their A-list, thus lines 11 and 1 have z_3 and z_4 on their P-list (and line 1 still has z_3 and z_4 also on its P-list). Lines 19 and 20 have lines 12 and 1 on their A-list, thus lines 12 and 1 have z_5 and z_6 on their P-list. Note, however, that z_5 and

z_4 are *not* on line 11's P-list, nor are z_7 and z_4 on line 12's P-list. Therefore line 11 can be UIed to z_3 and z_4, while line 12 can be UIed to z_7 and a_4.

21. $(Ez)(Aw)((Fz_1\&Gz_3)\rightarrow(Hz\&Iw))$ 11,UI

22. $(Ez)(Aw)((Fz_1\&Gz_4)\rightarrow(Hz\&Iw))$ 11,UI

23. $(Ez)(Aw)((Fz_2\&Gz_7)\rightarrow(Hz\&Iw))$ 12,UI

24. $(Ez)(Aw)((Fz_2\&Gz_4)\rightarrow(Hz\&Iw))$ 12,UI

This yields four new EIs to new variables

25. $(Aw)((Fz_1\&Gz_3)\rightarrow(Hz_3\&Iw))$ 21,EI

26. $(Aw)((Fz_1\&Gz_4)\rightarrow(Hz_2\&Iw))$ 22,EI

27. $(Aw)((Fz_2\&Gz_7)\rightarrow(Hz_1\&Iw))$ 23,EI

28. $(Aw)((Fz_2\&Gz_4)\rightarrow(Hz_2\&Iw))$ 21,EI

Since lines 25 and 26 have lines 11 and 1 on their A-list, z_3 and z_2 are on 11's and 1's P-list. And since lines 27 and 28 have lines 12 and 1 on their A-list, z_1 and z_2 are on 12's and 1's P-list. Note, however, that z_1 and z_2 are *not* on line 11's P-list, nor are z_3 and z_2 on line 12's P-list. This gives rise to more UIs on these lines; and by EI, to more new variables in the proof and hence to more UIs.

The problem here is that lines 11 and 12 should not have been UIed to z_7, z_4 and z_3, z_4 respectively. (I.e., the lines 21-24 should not have been allowed.) Rather, the proof should have proceeded

21. $(Fz_1\&Gz_3)\rightarrow(Hz_4\&Iz_7)$ 18,UI (or any variable for w)

22. $Hz_4\&Iz_7$ 7,21MP

23. Hz_4 22,S

24. $(Ex)Hx$ 23,EG

which will cancel goals up through line 5, and then the goal

25. show (Aw)Iw

would be added and proved similarly.

So it appears that when a universally quantified line comes by UI from another line, then not only is the latter on the former's A-list, but also the former should be on the latter's A-list. Here, lines 11 and 12 are universally quantified lines which come by UI from line 1, so not only is line 1 on lines 11 and 12's A-list, but also lines 11 and 12 should be on 1's A-list. That is, the universally quantified children of a universally quantified parent should be ancestors of that parent (in addition to the reverse).

The mechanism for constructing A-lists was suitably altered and a proof in Appendix I shows that this problem can be solved using it.

H. The UI/EI Problem, IV

I had thought that with the method indicated in the last section the UI/EI problem had been solved -- or at least solved to the extent it can be solved given the undecidability of first order logic. (The undecidable classes of formulae in first order logic are those whose prenex normal form has an existential quantifier in the scope of universal ones. And wasn't the repair indicated in the last section sufficient to handle existential quantifiers buried arbitrarily deeply inside universal

quantifiers? The problem brought out and solved in UI/EI, II only handled a depth of embedding of one, but in the last section this was generalized to any depth.) Indeed, that method can prove all the quantifier theorems from Chapter III of Kalish & Montague, and almost all those from Chapter IV. The ones it cannot prove run up against the fourth stage of the UI/EI problem.

Unfortunately, the way indicated in the last few sections is not the only manner that the infinite sequences of UIs-EIs can be brought about. For example, there might be two premises

1. (Ax)(Ey)Fxy
2. (Ax)(Ey)Gxy

(or these might have resulted from the assumption of a conjunctive antecedent). In these cases, 1 and 2 will have a null A-list. If z_8 is already in the proof, they will be instantiated to

3. (Ey)Fz_8y
4. (Ey)Gz_8y

which gives rise to EIs to new variables

5. Fz_8z_8
6. Gz_8z_7

where z_8 is on line 1's P-list and z_7 is on line 2's P-list. But since neither is z_8 on line 2's P-list nor z_7 on line 1's P-list, we can do more UIs to get

7. (Ey)Fz_7y
8. (Ey)Gz_8y

which leads to new EIs with new variables

9. Fz_7z_6

10. Fz_8z_5

where z_6 is on line 1's P-list and z_5 is on line 2's P-list, but not the reverse. Hence we can do more UIs followed by EIs, etc.

It seems here that once a variable is on one of the P-lists it ought to be on all the P-lists. But this is too strong, for if *no* variable is on a formula's P-list, a UI ought to be allowed to any variable, regardless of whether it is on someone else's P-list. That is, "innocent" universally quantified lines (with no P-list) should not be restricted in what they can do; only those universally quantified lines that have shown themselves to be "untrustworthy" (have something on their P-list) should be suspect.

So a radical restructuring was done. A-lists were constructed as before, but rather than have a separate P-list for each formula, one common P-list was maintained. And associated with each (universally quantified) formula was a boolean that indicated whether it had ever contributed to the common P-list. If it had, then no UI to any member of that list was permitted as a value of a UI (until it came time to do a FINDUIPROHIB).

I. The Logic of Set Theory

Some proofs in set theory can be formulated directly within first order logic without identity. We shall do this by allowing 'F^2' to stand for "is a member of". Russell's paradox can be put "there is no set which contains exactly those sets that are not members of themselves." This can be translated as:

$\neg(Ex)(Ay)(Fyx \leftrightarrow \neg Fyy)$

THINKER proves this quite easily (see Appendix I). Since "the Russell set" cannot exist, it follows that if there is a set of things all of whose members *are* members of themselves ("the anti-Russell set"), then not every set can have a complement. I.e.,

$(Ey)(Ax)(Fxy \leftrightarrow Fxx) \rightarrow \neg(Ax)(Ey)(Az)(Fxy \leftrightarrow \neg Fzx)$

Modern set theory replaces the unrestricted comprehension axiom (that every property determines a set) with a restricted version: given a set z, there is a set all of whose members are drawn from z and which satisfy some property. Now, if there were a universal set, then the Russell set could be formed, *per impossible*. So given the restricted comprehension axiom there is no universal set.

$(Az)(Ey)(Ax)(Fxy \leftrightarrow (Fxz \& \neg Fxx)) \rightarrow \neg(Ez)(Ax)Fxz$

Next, call a set x *circular* if it is a member of a set z which in turn is a member of x. Intuitively, all the sets are non-circular, but if we could pick out the class of the non-circular sets we could thereby pick out the universal set. Hence there can be no class consisting of exactly the

non-circular sets

$\neg(Ey)(Ax)(Fxy\leftrightarrow\neg(Ez)(Fxz\&Fzx))$

A final set theoretic problem is to prove that set identity is symmetric, given the definition of set identity in terms of having all the same members. Again let F stand for "is a member of" and let E: *a* is identical to *b*.

$(Ax)(Ay)(Exy\leftrightarrow(Az)(Fzx\leftrightarrow Fzy))\vdash(Au)(Av)(Euv\leftrightarrow Evu)$

This problem is reported by de Champeaux (1979: 195) as being unsolvable by his system. THINKER proves all these theorems handily. The proofs are listed in Appendix I.

J. Andrew's Challenge

According to de Champeaux (1979: 196), Peter Andrews posed the following problem at the fourth workshop on automated deduction at Austin, Texas, Feb. 1979.[**]

$[(Ex)(Ay)(Px\leftrightarrow Py)\leftrightarrow((Ex)Qx\leftrightarrow(Ay)Py)]\leftrightarrow$

$[(Ex)(Ay)(Qx\leftrightarrow Qy)\leftrightarrow((Ex)Px\leftrightarrow(Ay)Qy)]$

THINKER breaks this problem into six subproblems:

A. cases where antecedent is

$[(Ex)(Ay)(Px\leftrightarrow Py)\leftrightarrow((Ex)Qx\leftrightarrow(Ay)Py)]$

a. and the sub-antecedent is $(Ex)(Ay)(Qx\leftrightarrow Qy)$

i) show $(Ex)Px\rightarrow(Ay)Qy$

ii) show $(Ay)Qy\rightarrow(Ex)Px$

[**]The problem as reported by de Champeaux contains a number of apparently typographical errors. There are so many of them that it makes one think his theorem prover has not actually proved it, contrary to his claim. (And there is also the fact that his system couldn't prove the set identity problem of the last section which is considerably easier.)

b. and the sub-antecedent is (Ex)Px↔(Ay)Qy

i) show (Ex)(Ay)(Qx↔Qy)

B. cases where antecedent is

(Ex)(Ay)(Qx↔Qy)↔((Ex)Px↔(Ay)Qy)

a. and the sub-antecedent is (Ex)(Ay)(Px↔Py)

i) show (Ex)Qx→(Ay)Py

ii) show (Ay)Py→(Ex)Qx

b. and the sub-antecedent is (Ex)Qx↔(Ay)Py

i) show (Ex)(Ay)(Px↔Py)

Let us just take a look at one of these, problem Aai. THINKER assumes the main and subsidiary antecedents

1. (Ex)(Ay)(Px↔Py)↔((Ex)Qx↔(Ay)Py)

2. (Ex)(Ay)(Qx↔Qy)

3. (Ex)Px

and wants to show that everything is a Q. This obviously follows by the following chain of reasoning.

2 says that either everything is a Q or nothing is. If the former, THINKER is done. So we need but show that the second alternative is impossible. The left hand side (LHS) of 1 says that either everything is a P or nothing is. Given 3, if the LHS is true, it follows that everything is. Hence if the LHS is true, then (Ex)Qx; so the "second alternative" is impossible. Now, what if the LHS is false, i.e., some but not all are P's? In that case the RHS must also be false, so exactly one of (Ex)Qx and (Ay)Py is false. But given that not all are P's, it follows

that the one which is false is (Ay)Py. So (Ex)Qx is true, and again the "second alternative" is impossible.

Similar reasoning holds for all the cases. In Appendix I there are proofs of each of the six subproblems. (The entire problem is too long to be conveniently presented as a whole).

K. Schubert's Steamroller

In 1978, Lenhart Schubert presented the following problem to J. Siekmann of Universität Karlsruhe to test the graph-theoretic resolution prover of Siekmann and his associates. (See Chapter III for discussion of graph-theoretic resolution provers). This prover appears to be among the most advanced theorem prover in existence, but was unable to prove this argument.[4'] An English version of the argument is:

> Wolves, foxes, birds, caterpillars, and snails are animals, and there are some of each of them. Also there are some grains, and grains are plants. Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants. Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which in turn are much smaller than wolves. Wolves do not like to eat foxes or grains, while birds like to eat caterpillars but not snails. Caterpillars and snails like to eat some plants. Therefore there is an animal that likes to eat a grain-eating animal.

Symbolized, using the following abbreviations

[4']It is not known what the current status of the Karlsruhe theorem prover is.

P_0: *a* is an animal
P_1: *a* is a wolf
P_2: *a* is a fox
P_3: *a* is a bird
P_4: *a* is a caterpillar
P_5: *a* is a snail
Q_0: *a* is a plant
Q_1: *a* is a grain
S: *a* is much smaller than *b*
R: *a* likes to eat *b*

the argument becomes

$$(Ax)(P_1x \rightarrow P_0x) \;\&\; (Ex)P_1x$$
$$(Ax)(P_2x \rightarrow P_0x) \;\&\; (Ex)P_2x$$
$$(Ax)(P_3x \rightarrow P_0x) \;\&\; (Ex)P_3x$$
$$(Ax)(P_4x \rightarrow P_0x) \;\&\; (Ex)P_4x$$
$$(Ax)(P_5x \rightarrow P_0x) \;\&\; (Ex)P_5x$$
$$(Ex)Q_1x \;\&\; (Ax)(Q_1x \rightarrow Q_0x)$$
$$(Ax)(P_0x \rightarrow [(Ay)(Q_0y \rightarrow Rxy) + (Ay)((P_0y \;\&\; Syx \;\&\; (Ez)(Q_0 \;\&\; Ryz)) \rightarrow Rxy)])$$
$$(Ax)(Ay)((P_3y \;\&\; (P_5x + P_4x)) \rightarrow Sxy)$$
$$(Ax)(Ay)((P_3x \;\&\; P_2y) \rightarrow Sxy)$$
$$(Ax)(Ay)((P_2x \;\&\; P_1y) \rightarrow Sxy)$$
$$(Ax)(Ay)[(P_1x \;\&\; (P_2y + Q_1y)) \rightarrow \neg Rxy]$$
$$(Ax)(Ay)((P_3x \;\&\; P_4y) \rightarrow Rxy)$$
$$(Ax)(Ay)((P_3x \;\&\; P_5y) \rightarrow \neg Rxy)$$
$$(Ax)((P_4x + P_5x) \rightarrow (Ey)(Q_0y \;\&\; Rxy))$$
$$\vdash (Ex)(Ey)(P_0x \;\&\; P_0y \;\&\; (Ez)(Q_1z \;\&\; Ryz \;\&\; Rxy))$$

That the argument is valid can be seen by this reasoning:

> First, let's try to find a grain-eating animal. According to a premise, it's not a wolf. But since grains are plants, it follows that wolves do not like to eat all plants. So, by a premise, they must like to eat all animals much smaller than themselves that like to eat some plants. Since they also don't like to eat foxes, it follows that foxes do not eat any plants, and hence don't eat any grains. As far as caterpillars and snails go, all we are told is that they like to eat some plants; but we don't know about grains, and we can't figure it out since we are not told whether any animals are smaller than caterpillars or snails. This leaves the birds, and sure enough, since they do not like to eat snails (which like to eat some plants) it follows that they must like to eat all plants. So now we have to find an animal that likes to eat birds. Is it the wolf? Well, since wolves don't like to eat all plants, it

follows that they like to eat all animals much smaller than themselves that like to eat some plants. Are birds such animals? Certainly birds like to eat some plants, but are they much smaller than wolves? We can't tell. All we know is that birds are much smaller than foxes, which are much smaller than wolves; but we are not given the transitivity of "much smaller than". So we cannot prove that wolves like to eat birds. Is it the fox? Well, we've already proved that foxes do not like to eat any plants, so they must like to eat all animals much smaller than themselves which do like to eat some plants. Again, a bird is an animal that likes to eat some plants, and we are also given that birds are much smaller than foxes. Thus foxes like to eat birds. Since there are foxes and birds, it follows that there is an animal which likes to eat a grain-eating animal. QED

L. Can THINKER Prove the Steamroller?

THINKER can prove the Steamroller, at least in theory. However, due to time and space limitations it has not yet succeeded. It is of some interest to see why it can and why it hasn't, for this failure sheds some light on a direction to further develop THINKER in the future. (Such a development is discussed in the next Chapter).

First let's look at some theory about how proofs develop in Kalish & Montague. I wish to show that even when a provable (sub)proof is "started incorrectly", it can nonetheless be completed. For example, it is already built into the "proof completion" rules that

```
show (P→Q)
  P          ASSUME
  S
  ¬S
```

allows one to box and cancel, even though it was "started incorrectly" by assuming an antecedent of the conditional

whereas it finished by finding a contradiction. The reason for allowing this is because, given such a sequence of lines, there is another proof which "starts in the same mode that it ends." For the above, we have

```
* show (P→Q)
|    P              ASSUME
|    S
|    ¬S
|   * show Q
|   |  ¬Q          ASSUME
|   |  S           Repeat
|   |  ¬S          Repeat
```

the outside block of which starts by assuming an antecedent and ends by proving the consequent, while the inside block starts by assuming the negation and ends by finding a contradiction. All combinations of starting and ending proofs in Kalish & Montague have the property that if a subproof is started in one way and ended in another, then there is a proof which starts in the same manner it ends. Since this is so, Kalish and Montague merely allow any combination of "starts and ends."

In a similar vein, suppose an indirect subproof for a provable formula has been started, so that there is a contradiction lurking somewhere. And now suppose that an "extraneous" show line is added to the proof. Finally, suppose that we now discover the contradiction. Finding it of course only allows boxing and cancelling up to the "extraneous" show line. (Recall that one cannot enclose an uncancelled show in a box). Can we recover from the mistake of adding the extraneous show line? The answer is yes.

We suppose that an indirect proof has been started and that we could derive a contradiction from this but have instead entered the extraneous "show ø" line.

```
  .
  .
  .
show P
 ¬P        --start the indirect proof
  .
  .
  .
 show ø     --extraneous show line
  .
  .
 Q
 ¬Q         --derive a contradiction
```

What should be done now is box and cancel the ø show line, and then re-derive the contradiction. By hypothesis this can be done: since Show ø is superfluous, nothing depends on it (or on any assumptions it generates) to find Q and ¬Q. If it *were* required, it wouldn't be superfluous. All that is needed is that the contradiction be lurking prior to the Show ø line.

. A simple generalization of this is when a show line is required but the wrong one is written down. In the above example, suppose we need to write down Show ∮, but instead write Show ø. This too can be remedied

```
  .
  .
  .
show P
 ¬P        --start the indirect proof
  .
  .
  .
 show ø     --extraneous show line
  .
```

```
.
.
show ψ  --correct show line
```

Now by hypothesis Show ψ is provable and what is needed in the subproof of Show P. (It is needed because, for example, it contradicts some line between Show P and Show ø, or because P is a conditional and ψ is its consequent, etc.) In any of these cases, the proof could continue by cancelling Show ψ, repeating the reason it was necessary (e.g., the line it will contradict), using this to cancel Show ø, and then re-setting the goal Show ψ, this time not within the Show ø subproof. We now prove ψ and this time it will cancel Show P. Suppose the reason to set Show ψ is that it will contradict ¬ψ which is a line between Show P and Show ø. We then have the lines

```
.
.
.
*show P
|  ¬P
|  .
|  .
|  ¬ψ
|  .
|  .
|  *show ø     --superfluous
|  |  .
|  |  .
|  |  *show ψ --provable, by hypothesis
|  |  |  .
|  |  |  .
|  |  |  .
|  |  ¬ψ       --Repeat
|  *show ψ     --reset correct goal
|  |  .
|  |  .
|  |  .
```

In other words, if Show ø is truly superfluous, and if the

correct show line (here: Show ψ) can be found, then even though we started incorrectly by writing Show ϕ, this can be overcome. One does it by proving (the correct) Show ψ twice -- once in the scope of (the incorrect) Show ϕ subproof and once afterwards. This is at the cost of some extra writing, but can always be done. Furthermore, no matter how many superfluous show lines are written, so long as the correct one eventually comes up, this strategy will eventually yield a proof.

This sort of situation happens in THINKER's attempted proof of the Schubert Steamroller. There are six existential premises here, so THINKER does six EIs (to six different variables, call them a, b, c, d, e, f). It now does a number of UIs on the other premises and some number of MPs. Since these are done to *all* six of our variables, there are quite a large number of lines left, differing from one another only in what variables they contain. In particular, note that the following types of lines are generated. From assuming the negation of the conclusion, doing QNs and UIs, we have

(1) $\neg(P_0\alpha \;\&\; P_0\beta \;\&\; Q_1\gamma \;\&\; R\beta\gamma \;\&\; R\alpha\beta)$

for all combinations of a, b, c, d, e, f substituted for α, β, and γ. (So an indirect proof has been started and there is a contradiction lurking somewhere). The other type of lines which are of interest have have the form

(2) $(Ay)(Q_0y \rightarrow R\alpha y) + (Ay)((P_0y \& Sy\alpha \& (Ez)(Q_0z \& Ryz)) \rightarrow R\alpha y)$

from a premise (for each of our six variables replacing α).

As it turns out, the most efficient strategy is to set the unnegated correct instance of (1) as a goal and then as a subgoal set the negation of a disjunct from the correct instance of (2), using the CHAINING strategy. In particular, if b is the variable generated by EI from $(Ex)P_2x$ (the fox) and c is the variable generated by EI from $(Ex)P_3x$ (the bird) and f is the variable generated from $(Ex)Q_1x$ (the grain), we want to set

show $(P_0b \ \& \ P_0c \ \& \ Q_1f \ \& \ Rcf \ \& \ Rbc)$

as a goal. When it is proved, it will cancel the higher goal by contradicting the appropriate instance of (1). The SEARCHNEGS strategy finds negations of formulae like (1), but unfortunately it does not always get the correct instance first. However, as I showed above, even if it doesn't get the correct instance first, as long as it eventually gets it and proves it, the proof will succeed.

Will THINKER ever get the correct instance? And if it does, will it then (after proving the superfluous subgoal) turn around and re-set the correct instance as a goal if it needs it? The answer to both questions is yes, but it will take a long time. As long as the proof has not been completed, and so long as the unnegated formula is not already a goal, FINDNEGS simply keeps adding these formulae to the goal stack until there are no more to be added. So eventually the correct one will be added. Suppose now that the correct instance is added and proved, and thus the superfluous instance is proved. It turns out that in the

case of FINDNEGS there is no need to re-set the goal. For, having proved even an irrelevant instance of (1), it can cancel the next higher (irrelevant) goal and so on, up to the point where the indirect proof was begun. This is so because each of these goals, whether relevant or not, was set on the grounds that proving it would allow an immediate cancellation of the next higher goal (due to a contradiction). Thus in the case that the irrelevant goals were set by SEARCHNEGS, the proof of one (the correct, but embedded) goal suffices to prove them all.

The matter is a bit complicated in the Steamroller case, because there are so many incorrect goals. To see what will happen, consider the case of two "wrong" and one "correct" instance of (1), where the two wrong goals are set first. Call the wrong instances $\neg I_1$ and $\neg I_2$, and the correct instance $\neg I_3$. THINKER would proceed as follows

.
.
.
$\neg I_1$
$\neg I_2$
$\neg I_3$
.
.
*show I_1
.
.
*show I_2
.
.
*show I_3

We now suppose it can prove I_3 and thereby cancel I_2, and then THINKER will consider whether it needs to re-set I_3 as a goal. It doesn't need to:

We see here that the next higher show can now be boxed, due to the presence of $\neg I_1$ and I_1. With the SEARCHNEGS strategy *any* of the instances proved will allow continuous cancellation of goals "upward". All that's required in the present case is to prove I_3, no matter how far embedded. How far is it embedded? Well, given six variables, and taking them three at a time (with repetition) as indicated by formula (1), we have 6^3 (= 216) instances of (1). One of these is the correct instance.

As it turns out, however, to prove the correct instance we need to do CHAINING by using formula (2). We need to set the negation of a disjunct of the correct instance of (2) as a goal. The actual subproof here is rather tedious, but in outline it goes like this.

```
m. *show (P.b&P.c&Q.f&Rcf&Rbc) --correct (1)
   |  .
   |  .
   |  .
n. |  *show ¬(Ay)(Q.y→Rby)      --correct disjunct of (2)
   |     (Ay)(Q.y→Rby)          --ASSUME
   |     .
   |     .
p. |  (Ay)(P.&Syb&(Ez)(Q.z&Ryz)→Rby) --lines n,(2) MTP
   |  .
   |  .
   |  .
```

Of course, there are doubtless superfluous show lines between m and n -- all those incorrect instances of (2) that THINKER finds before n. And one might ask, since the case is different than it was before with SEARCHNEGS, whether it is always guaranteed that an eventual embedded box and cancel always allows upward cancellation. After all, one asks, on each level we merely use the embedded cancellation to generate an MTP. What guarantee is there that the result of MTP on a superfluous instance will lead to a cancellation of a higher show line? Suppose then we have such a case -- suppose there is one superfluous instance of the (2) show line before we hit upon the correct instance.[43] Let I be the correct instance of (1), W be the wrong instance of (2), C be the correct instance of (2), and W' and C' be the results of MTP with W or C on the appropriate instance of (2). We then have

```
m.  show I          --correct (1)
    .
    .
    .
m'. show W          --incorrect (2)
    .
```

---

[43]There are a total of six instances of (2).

```
n.    *show C          --correct (2)
      |  .
      |  .
p.    C'               --n, (2) MTP
      .
      .
      .
```

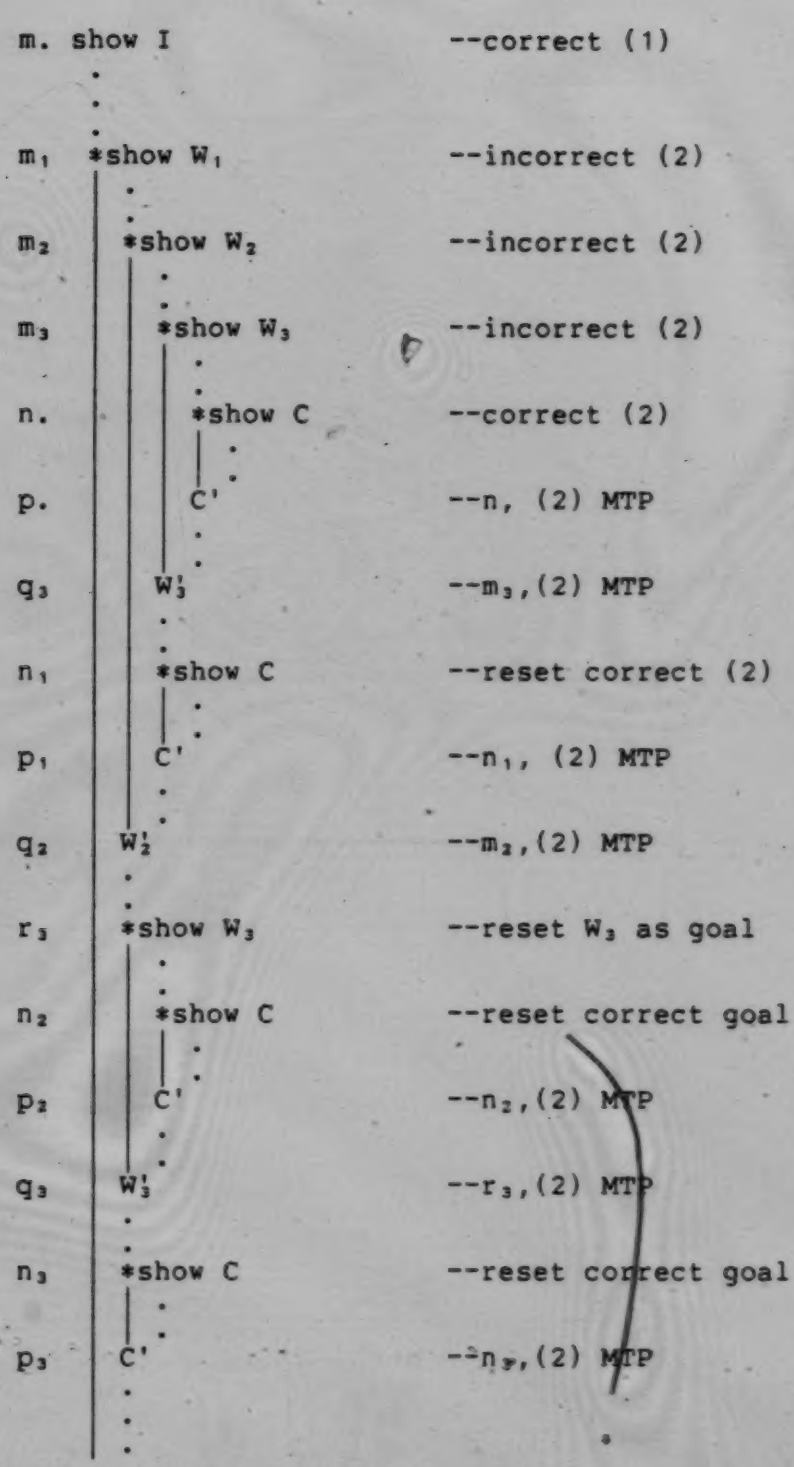
Now, by hypothesis the proving of the correct instance of (2) leads to a MTP (our line p) which would eventually cancel line m, that which we wish to prove. (Because an indirect proof has been started). Here we use it to cancel line m' instead. Proving line m' now allows us to do an MTP with the appropriate instance of (2).

```
m.  show I                   --correct (1)
    .
    .
    .
m'. *show W                  --incorrect (2)
    |  .
    |  .
    |  .
n.  |  *show C               --correct (2)
    |  |  .
    |  |  .
p.  |  C'                    --n, (2) MTP
    |  .
    |  .
    |  .
p'. W'                       --n, (2) MTP
```

Now, is there any guarantee that from p' we can eventually cancel m? Yes: all that is required is to re-set C as a goal. This *can* be done since it is no longer on the goal stack, and (unless THINKER succeeds without it) it *will* be done since THINKER eventually places all possible CHAINING goals on the goal stack. And by hypothesis the proof of C allows a cancellation of I.

As can be seen, this process can become very messy indeed. Unlike the case with SEARCHNEGS, where the cancellation of any goal allows the cancellation of them all (because they were all generated as negations of lines currently antecedent), in CHAINING here the cancellation of a goal merely allows the proof to proceed further. And if the "further proceeding" is up a blind alley, THINKER needs to return to the embedded correct goal for help; it needs to re-set that goal outside the scope of the superfluous goal, prove it again, and use this to prove the next higher goal. Given that THINKER finds these goals by going (deterministically) around the +-ring looking for formulae of the form $(\phi+\psi)$, it will always re-set its goals in the same order.

This is unfortunate if there are a number of superfluous goals set. Consider the following where instead of one superfluous goal there are three. (I use W_1, W_2, W_3 for these, and W_1' etc. for the result of performing MTP with the appropriate instance of (2). The example is on the next page.) The preceding discussion explains how line m_2 gets cancelled: the n...p correct instance proof cancels m_3, and then is repeated as $n_1...p_1$, which cancels m_2. m_2 is used with its corresponding instance of (2) to generate q_2. However, m_2 was superfluous, so q_2 will not yield a contradiction needed to cancel the next higher goal. THINKER needs to reset C as a goal. But before it can do that, it must set (the superfluous) W_3 as a goal, since W_3 was


m. show I --correct (1)
m1 *show W1 --incorrect (2)
m2 *show W2 --incorrect (2)
m3 *show W3 --incorrect (2)
n. *show C --correct (2)
p. C' --n, (2) MTP
q3 W3' --m3,(2) MTP
n1 *show C --reset correct (2)
p1 C' --n1, (2) MTP
q2 W2' --m2,(2) MTP
r3 *show W3 --reset W3 as goal
n2 *show C --reset correct goal
p2 C' --n2,(2) MTP
q3 W3' --r3,(2) MTP
n3 *show C --reset correct goal
p3 C' --n3,(2) MTP

assumed to "come before" C in the +-ring and is also not antecedent at this stage. So now the proof continues with W_3

reset as a goal, and then C is reset as a goal. Of course, having the correct goal (line n_2) within the scope of an incorrect goal (line q_2) means that the correct goal has to be reset once again outside its scope. And now the (correct) proof between $n_2 \ldots p_2$ allows cancellation of the next higher goal, in this case W_1. This is as far as has been illustrated in the schematic proof: line m_1 has been cancelled. Line m_1, however, is another superfluous goal. Thus in order to cancel line m, as we wish, we need to reset C as a goal. But we cannot do this until W_2 and W_3 are set as goals.

To make a long story shorter, now that W_1 is cancelled, W_2 is set as a goal at the same level of embedding as W_1. This sets W_3 and then C as subgoals, eventually cancelling W_2, essentially by redoing the part of the proof between lines m_2 and q_2. However, W_2 is also a superfluous goal, so the "next" goal (W_3) is set as a goal at the same level of embedding as W_1 and W_2. This in turn sets C as a subgoal, eventually cancelling W_3, by redoing the proof between r_3 and q_3. But W_3 is also superfluous, so the next goal, C, has to be set at the same level of embedding as W_1, W_2, and W_3. When this succeeds, THINKER can finally cancel line m, the correct instance of (1).

In general then, CHAINING finds goals in a certain order. If superfluous goals are found first, an embedding of the goals takes place until the correct one is found. Finding the correct one allows a cancellation of the most

deep incorrect one, and then a repeat of the correct one, to cancel the next higher incorrect one. But then the deeper incorrect one is reset, embedding the correct one, proving it, resetting the correct one and proving the deepest incorrect one, and resetting the correct one and proving the next deepest incorrect one. But then the embedded incorrect ones need to be reset. This continues until all goals up to and including the correct one (in the +-ring ordering) are cancelled at the same level of embedding.

Suppose the subproof of C plus the MTP and finding the contradiction takes i steps, and suppose there are k "pointless" steps in each superfluous part (that is, steps not directly involved in an immediate subproof), and that there are n superfluous subgoals before C in the +-ring. How long will it take to prove the higher goal? Well, to prove the most deeply embedded incorrect instance takes the proof of C together with whatever superfluous steps are required, i.e., i+k steps. The next higher one has embedded that subproof plus a subproof of C plus the pointless steps, i.e., (i+k)+(i+k). The third deepest embedding has that one plus an embedding of the subproof of the second deepest plus the subproof of C plus the pointless steps. In general, to prove the very first superfluous goal takes

SUM(j=1→j=n) [j(i+k)]

steps. But there are n of these to prove, each with one less embedding, before we come to C being at the same level as the first. I.e., to finally prove what is required

necessitates

SUM(m=1→m=n) SUM(j=1→j=m) [j(i+k)]

total steps.

The upshot of this is that THINKER in theory will eventually find a proof of the Steamroller. But it will not be found in any reasonable time. I said before that there were six instances of (2). While this is true, CHAINING is a more general procedure than just looking for (ø+ɸ) lines: it also looks for conditionals and sets their antecedents as goals or the negation of their consequents. In the Steamroller there are an incredibly large number of conditionals. It is therefore an extremely large problem. THINKER was given the problem and allowed to run for one minute CPU time on an Amdahl 470 V/8. It generated 2850 lines of proof. The proof was structured correctly, as indicated above -- the proper instance of (1) was finally set as a goal (at line 350), and shortly thereafter[43] CHAINING was called and started setting superfluous subgoals. The first of these had not been proved by line 2850, although some of the most deeply embedded ones were proved. It seems clear therefore that THINKER *can* prove the Steamroller, and would do so given more time.

---

[43] Of course, finding the proper instance of (1) does not mean that SEARCHNEGS will stop setting goals. Since the proper instance requires CHAINING to prove it, and since CHAINING is called after all SEARCHNEGS are found, there are many more SEARCHNEG show lines generated before the first of the CHAINING show lines is generated. The first CHAINING show line is at line 776.

VIII. SOME DIRECTIONS FOR THE FUTURE

A. Improvements to the Proof Strategy

There are two areas in which THINKER's strategies are not driven by any assurance that the result of applying the strategy will aid in getting "closer" to the goal. One is in the FIND-rules, where THINKER applies the rules of inference to all antecedent lines regardless of whether it is guaranteed that they will be useful lines.[4] It may be that there is some way to recognize "usefulness" of the possible applications of the rules in FIND, and to do the ones judged "more useful" first. But inspection of the sample proofs shows that the unbridled application of the FIND strategy does not unduly lengthen proofs.

The other place where inference is not goal-driven is in the TRYCHAINING strategy.[5] Here THINKER merely finds some conditional (or disjunction) amongst the antecedent lines for which the consequent (or other disjunct) is not

---

[4] Of course, THINKER only applies "structure-reducing rules" in FIND -- those rules which make their conclusion "less complex" than the most complex premise of the rule. E.g., MP (from ϕ and $(\phi \rightarrow \psi)$ infer ψ) is such a rule. Thus the FIND strategy will terminate in a reasonably short time, depending only on the number and complexity of antecedent lines, for once an antecedent line is used it is not again to be used in the same rule. Except for FINDUI, which is handled separately as discussed in Chapter VII, there is no rule which is either used again or increases structure. (So for example the rule Adj (from ϕ and ψ infer $(\phi \& \psi)$) is not a part of the FIND strategy.

[5] The FINDNEGS strategy, on the other hand, *is* goal-driven. As soon as one of the generated show lines is proved, the proof is completed up to a higher show level. In the discussion of the Steamroller of Chapter VII it is rather the TRYCHAINING strategy which is unfocused.

also an antecedent line, and sets an appropriate goal (the antecedent of the conditional, or the negation of one of the disjuncts). This strategy is *useful* in the sense that, if the goal is proved, then a new MP (or MTP) can be performed. But it is not *immediately useful* in the sense that the new MP (or MTP) will help the proof of the next higher goal.

It seems clear that it would be desirable to first look for immediately useful CHAININGs before trying all CHAININGs. One type of immediately useful chaining is BACKCHAINING: if ψ is the most recent goal, look for $(\varphi \rightarrow \psi)$ lines (or $(\neg\varphi + \psi)$ lines) in the antecedent lines and if found set as new goal the formula which is the token of φ. If this can be proved then the most recent goal can be immediately cancelled. Somewhat more generally, if ψ is *any* previous goal, look for $(\varphi \rightarrow \psi)$ lines (or $(\varphi + \psi)$ lines) amongst the antecedent lines and, if found, set the formula corresponding to φ as a goal. If this can be proved, then ψ will be a new antecedent line and (perhaps after some small amount of further processing) all goals up to and including the ψ goal can be cancelled.

These alterations are extremely easy to implement, and are currently being worked on. A harder-to-implement generalization would be the following. For all goals ψ, look for $(\phi \rightarrow \theta)$ antecedent lines (or the equivalent disjunction) such that θ allows for a one step proof of ψ. If found, set ϕ as a new goal. This would aid proofs immensely, but it seems that THINKER would have to produce little subproofs to

check whether one step on Θ would yield ϕ. Such attempts would not occur within the main proof matrix and would amount to "trying out a short proof to see whether it works, and if not discarding the attempt". One of the design goals has been not to allow THINKER to "throw away" proofs once started. Without this constraint, the actual work done by THINKER remains hidden -- we do not know what actual proof steps have been taken. In the present design, everything that THINKER does can be inspected by looking at the final proof it produces.

Even with the addition of BACKCHAIN, there will remain proofs that are very difficult for THINKER, for example, the Steamroller. As I discussed in Chapter VII, the problem here is that THINKER sets up an incredibly large number of goals in the CHAINING strategy, and it is only some of them that are worthwhile (although these worthwhile goals need not be immediately useful in the sense required by BACKCHAIN). What we need is to find which CHAIN to start with; but since there may not be any overt clue (e.g., even the one step strategy suggested in the previous paragraph may not yield enough information) as to which CHAIN to start with, it seems that we need to check them all. A suggestion that comes to mind is to start concurrent (parallel) processes for each of the possible CHAINs. The first one to prove a goal generated by the CHAIN strategy sends a message to its parent process which in turn stops all the other child processes. Of course one of these concurrent processes might

itself spawn a wide range of child processes if it should come to the point where a new instance of CHAINING is called for. Information between the children of a given process probably should allow a process to continue until it adds a line to its proof matrix, and then pass control to the next sibling process. Probably also, no sibling should be allowed to add a new CHAINING goal until all siblings are ready to do so.

A number of design decisions have to be made before this is attempted. For one thing, there is the problem of sharing between parent process and child process such data structures as ANTELINES, but having separate proof matrices (since the parent does not know which child is going to "win"). A further problem is that SPITBOL does not support concurrent processing -- although the SNOBOL-based language ICON does. Further work on this area is planned for the near future.

B. Identity and Functions

Two obvious extensions to THINKER are to extend the system to handle identity and arbitrary function symbols. In fact, THINKER already recognizes identity-formulae as well-formed by the formation rule

If α and β are terms, then $\alpha=\beta$ is a formula

However, there are no heuristics involving identity and so the only identity-arguments that can be proved are those which do not essentially involve identity. The rules I would

propose are these:

$\vdash (Ax)x=x$ (Ref)

and

$\phi, \alpha=\beta \vdash \phi'$ (LL)

where ϕ' comes from ϕ by proper substitution of some occurrence of α in ϕ by β (α and β are terms).'' One needs, however, some strategies as to when to use these rules. As a first step, I propose to implement Ref by adding an additional check in FINDCONTRA: if a line is of the form $\neg\alpha=\alpha$, then introduce $\alpha=\alpha$ as a line (annotated 'Ref') and immediately box and cancel. LL is a harder rule to implement. One *could* follow the "blind" procedure FINDUIS and merely do all the substitutions that are available. In the end one may have to have such a strategy to fall back on, but one hopes that a more efficient and controlled use of LL is sometimes possible. For example, given that $\alpha=\beta$ is in the proof, one should look for the pairs $\phi\alpha$ and $\neg\phi\beta$ (where $\phi\alpha$ and $\phi\beta$ differ only in that one has an occurrence of α where the other has occurrences of β). With a somewhat fancier TEMPLATE mechanism than THINKER now has, such a strategy should be implementable. In a similar vein, given $\alpha=\beta$ we should look for cases of the form $\phi\alpha$, $(\phi\beta\rightarrow\psi)$ as an opportunity to do MP (and the like for the other rules).

''These are not the rules of Kalish & Montague. They give these two:

$\phi' \vdash (A\alpha)(\alpha=\beta\rightarrow\phi)$

$(A\alpha)(\alpha=\beta\rightarrow\phi) \vdash \phi'$

and present Ref and LL as derived rules. However, from Ref and LL, they in turn can be derived, and so the present rules are adequate.

In theory, arbitrary function symbols are easy to implement. One characterizes them as:

f_i^j are (i-place) function symbols

If f is an i-place function symbol, and $\alpha_1 \ldots \alpha_i$ are terms, then $f(\alpha_1, \ldots, \alpha_i)$ is a term.

And since these are terms, they operate in a proof exactly as any other term (variable or constant). The real problem is the large number of terms this will introduce into a proof and the concomitantly large number of possible UIs this gives rise to. With this large number of new terms, however, the identity problems re-emerge with a vengance, especially in the proposed "blind" identity substitution procedure.

These aspects, though tedious, could easily be grafted onto the current version of THINKER. With such additions, THINKER could be more easily compared to the published versions of other theorem provers. For, in these publications, the main attempt has been to prove certain theorems in elementary mathematics such as group theory, ring theory, and semi-group theory. (One adds the axioms of the theory as premises of the arguments). In Kalish & Montague, Chapters 8 - 11, this method is used to generate various extensions of the theory. In particular, the following theories are developed: the theory of commutative ordered fields, the theory of real numbers, the theory of convergence, differential calculus, and integral calculus. Given the ease with which various theorems of these theories

are proved in Kalish & Montague, it should not be surprising if THINKER, so augmented, also could prove interesting theorems in these fields.

C. Natural Language Processing and Other Areas

Any of the areas mentioned in Chapter I would be a suitable test ground for THINKER. However, the area I am interested in is natural language processing. In particular, I am interested in the style of grammar promoted by Gerald Gazdar and his associates (see Gazdar 1981, Sag 1981, Gazdar *et al* forthcoming; see also Schubert & Pelletier 1982 for further details). In this conception of grammar, the semantic component is a (typed) lambda calculus in the style of Montague (1970, 1973). I would like to allow THINKER to accept formulae that are well-formed in the lambda calculus and perform logical operations on them, i.e., construct proofs in the lambda calculus. Besides an expansion in the set of formulae that are recognized by THINKER, the lambda calculus employs one more rule of inference: lambda conversion. Since such a rule does no more than find an equivalent of a formula, this is extremely easy to program. Indeed, I have in mind that the parsing component of the natural language system will produce unconverted lambda expressions corresponding to English sentences. THINKER will perform the appropriate lambda conversions and arrive at more "natural" equivalent expressions. In the Montague theory, these expressions are then further reduced by means

of "meaning postulates" which find various consequences of the unreduced formulae. THINKER would naturally be called upon to perform these further reductions.

But all of this is for the future. For now I've gone as far as I can go.

IX. APPENDIX I: THINKER's PROOFS OF SOME THEOREMS

This Appendix gives the proofs generated by THINKER for a selection of problems presented to it. The selection comprises just those used to illustrate points made in the text concerning how proofs proceed and concerning comparisons with other theorem provers.

It should be emphasized that the proofs are exactly as produced by THINKER. There has been no postprocessing of any sort other than conversion of internal representation to printer format. (For example, the symbol '→' which is used on output is internally '>'. Similar remarks hold for sub- and superscripts). A problem is presented to THINKER by typing in

premise ((p→q)→(r&s))

prove (q→s)

for example. There is no preprocessing of formulae.'' The internal SPITBOL clock and statement count have been appended to these proofs here by hand. (It turned out to be very difficult to format that part of THINKER's output so as to stay within the required margins).

We start with some simple propositional logic problems.

1. ⊢ (¬¬p→p)
2. ⊢ (((p→q)→p)→p)
3. (q→r), (r→(p&q)), (p→(q+r)) ⊢ (p↔q)
4. ⊢ ((p→q)↔(¬q→¬p))
5. ⊢ (¬(p→q)→(q→p))

''Problems with no premises are just entered with "prove" plus formula.

6. ⊢ (p↔↔p)

7. ⊢ (p→¬¬¬p)

8. ⊢ (((p↔↔q)↔↔r)↔↔(p↔↔(q↔↔r)))

9. ⊢ (((p→q)→(p→r))→(p→(q→r)))

10. ⊢ ((¬p→q)→(¬q→p))

11. ⊢ (((p&q)→(p&¬q))→((¬p&q)→(¬p&¬q)))

(1) is the "hardest" theorem proved by the "new Logic Theorist". (5) is the "hardest" theorem proved by Siklóssy with a breadth first search. (10) is the theorem judged by Siklóssy to be "hardest" of the first 52 theorems of Whitehead & Russell (1910), and (9) his judgment of the "hardest" one of the first 62 theorems. (4) is a biconditional version of the "most difficult" problem proved by the "original Logic Theorist". (That is, the original Logic Theorist could only prove one direction of it: ((p→q)→(¬q→¬p)).) (7) is a problem of which it has been proved that the "original Logic Theorist" could not prove, and (11) is a problem that cannot be proved by unit resolution (nor therefore by input resolution). For details of these items, see Chapter VII Section A. (2) is known in logic as "Peirce's Law" after the 19th century American logician Charles Sanders Peirce. (3) illustrates THINKER's proving an argument with premises.

THINKER's proof of (6) is interesting. Most elementary students, in trying to prove any biconditional, will try to show each direction separately. Here however, both directions are the same. THINKER, unlike most elementary

students, notices this and does not bother to write a 'show' line for the "other" direction. (8), the associativity of $\leftrightarrow$, is the "hardest" of the propositional theorems THINKER has been asked to prove. (It is Theorem 95 of Kalish & Montague).

```
1 *show (¬¬p→p)
2 | ¬¬p            ASSUME
3 | p              2,DN
```

CPU time: 38 msec
Statements: 420

```
1 *show (((p→q)→p)→p)
2 | ((p→q)→p)              ASSUME
3 | *show p
4 | | ¬p                  ASSUME
5 | | ¬(p→q)              4,2,MT
6 | | *show (p→q)
7 | | | p                ASSUME
8 | | | ¬p               4,R
```

CPU time: 52 msec
Statements: 1766

```
 1 *show (p↔q)
 2 | *show (q→p)
 3 | | q                    ASSUME
 4 | | *show p
 5 | | | ¬p                 ASSUME
 6 | | | (q→r)              PREM
 7 | | | r                  6,3,MP
 8 | | | (r→(q&p))          PREM
 9 | | | (q&p)              8,7,MP
10 | | | p                  9,S
11 | *show (p→q)
12 | | p                    ASSUME
13 | | *show q
14 | | | ¬q                 ASSUME
15 | | | (p→(q+r))          PREM
16 | | | (q+r)              15,12,MP
17 | | | r                  14,16,MTP
18 | | | (r→(q&p))          PREM
19 | | | (q&p)              18,17,MP
20 | | | q                  19,S
21 | (p↔q)                  11,2,CB
```

CPU time: 64 msec
Statements: 3351

```
 1 *show ((p→q)↔(¬q→¬p))
 2 |  *show ((¬q→¬p)→(p→q))
 3 |  |  (¬q→¬p)                    ASSUME
 4 |  |  *show (p→q)
 5 |  |  |  p                      ASSUME
 6 |  |  |  q                      3,5,MT
 7 |  *show ((p→q)→(¬q→¬p))
 8 |  |  (p→q)                      ASSUME
 9 |  |  *show (¬q→¬p)
10 |  |  |  ¬q                     ASSUME
11 |  |  |  ¬p                     8,10,MT
12 |  ((p→q)↔(¬q→¬p))              7,2,CB
```

CPU time: 60 msec
Statements: 2264

```
1 *show (¬(p→q)→(q→p))
2 | ¬(p→q)                ASSUME
3 | *show (q→p)
4 | | q                  ASSUME
5 | | *show p
6 | | | ¬p               ASSUME
7 | | | *show (p→q)
8 | | | | q              4,R
9 | | | ¬(p→q)           2,R
```

CPU time: 53 msec.
Statements: 1826

```
1 *show (p↔p)
2 | *show (p→p)
3 | | p                ASSUME
4 | (p↔p)              2,2,CB

CPU time: 41 msec
Statements: 622
```

```
1 *show (p+¬¬¬p)
2 | ¬(p+¬¬¬p)          ASSUME
3 | *show p
4 | | ¬p               ASSUME
5 | | *show ¬¬¬p
6 | | | ¬¬¬p          4,DN
7 | | ¬(p+¬¬¬p)        2,R
8 | | (p+¬¬¬p)         5,ADD
9 | (p+¬¬¬p)           3,ADD
```

CPU time: 49 msec
Statements: 1625

```
1  *show (((p↔q)↔r)↔(p↔(q↔r)))
2  | *show ((p↔(q↔r))→((p↔q)↔r))
3  | | (p↔(q↔r))                                  ASSUME
4  | | *show ((p↔q)↔r)
5  | | | *show (r→(p↔q))
6  | | | | r                                     ASSUME
7  | | | | *show (p↔q)
8  | | | | | *show (q→p)
9  | | | | | | q                                 ASSUME
10 | | | | | | *show p
11 | | | | | | | ¬p                              ASSUME
12 | | | | | | | (p→(q↔r))                       3,BC
13 | | | | | | | ((q↔r)→p)                       3,BC
14 | | | | | | | ¬(q↔r)                          11,13,MT
15 | | | | | | | *show (q↔r)
16 | | | | | | | | *show (r→q)
17 | | | | | | | | | q                             9,R
18 | | | | | | | | *show (q→r)
19 | | | | | | | | | r                             6,R
20 | | | | | | | | (q↔r)                           18,16,CB
21 | | | | | *show (p→q)
22 | | | | | | p                                 ASSUME
23 | | | | | | *show q
24 | | | | | | | ¬q                              ASSUME
25 | | | | | | | (p→(q↔r))                       3,BC
26 | | | | | | | ((q↔r)→p)                       3,BC
27 | | | | | | | (q↔r)                           25,22,MP
28 | | | | | | | q                               27,BC,6,MP
29 | | | | | (p↔q)                                 21,8,CB
30 | | | | *show ((p↔q)→r)
31 | | | | | (p↔q)                                 ASSUME
32 | | | | | *show r
33 | | | | | | ¬r                                  ASSUME
34 | | | | | | (p→(q↔r))                         3,BC
35 | | | | | | ((q↔r)→p)                         3,BC
36 | | | | | | (p→q)                             31,BC
37 | | | | | | (q→p)                             31,BC
38 | | | | | | *show p
39 | | | | | | | ¬p                                ASSUME
40 | | | | | | | ¬(q↔r)                            39,35,MT
41 | | | | | | | ¬q                                39,37,MT
42 | | | | | | | *show (q↔r)
43 | | | | | | | | *show (r→q)
44 | | | | | | | | | r                               ASSUME
45 | | | | | | | | | ¬r                              33,R
46 | | | | | | | | *show (q→r)
47 | | | | | | | | | q                               ASSUME
48 | | | | | | | | | ¬q                              41,R
49 | | | | | | | | (q↔r)                             46,43,CB
50 | | | | | | (q↔r)                               34,38,MP
51 | | | | | | q                                   36,38,MP
52 | | | | | | r                                   50,BC,51,MP
53 | | | | ((p↔q)↔r)                                 30,5,CB
```

```
54  | *show (((p↔q)↔r)→(p↔(q↔r)))
55  | | ((p↔q)↔r)                          ASSUME
56  | | *show (p↔(q↔r))
57  | | | *show ((q↔r)→p)
58  | | | | (q↔r)                          ASSUME
59  | | | | *show p
60  | | | | | ¬p                           ASSUME
61  | | | | | ((p↔q)→r)                    55,BC
62  | | | | | (r→(p↔q))                    55,BC
63  | | | | | (q→r)                        58,BC
64  | | | | | (r→q)                        58,BC
65  | | | | | *show (p↔q)
66  | | | | | | *show (q→p)
67  | | | | | | | q                        ASSUME
68  | | | | | | | r                        63,67,MP
69  | | | | | | | (p↔q)                    62,68,MP
70  | | | | | | | (q→p)                    69,BC
71  | | | | | | *show (p→q)
72  | | | | | | | p                        ASSUME
73  | | | | | | | ¬p                       60,R
74  | | | | | | (p↔q)                      71,66,CB
75  | | | | | (p→q)                        65,BC
76  | | | | | (q→p)                        65,BC
77  | | | | | r                            61,65,MP
78  | | | | | q                            64,77,MP
79  | | | | | p                            78,76,MP
80  | | | *show (p→(q↔r))
81  | | | | p                              ASSUME
82  | | | | *show (q↔r)
83  | | | | | *show (r→q)
84  | | | | | | r                          ASSUME
85  | | | | | | *show q
86  | | | | | | | ¬q                       ASSUME
87  | | | | | | | ((p↔q)→r)                55,BC
88  | | | | | | | (r→(p↔q))                55,BC
89  | | | | | | | (p↔q)                    88,84,MP
90  | | | | | | | q                        89,BC,81,MP
91  | | | | | *show (q→r)
92  | | | | | | q                          ASSUME
93  | | | | | | *show r
94  | | | | | | | ¬r                       ASSUME
95  | | | | | | | ((p↔q)→r)                55,BC
96  | | | | | | | (r→(p↔q))                55,BC
97  | | | | | | | ¬(p↔q)                   94,95,MT
98  | | | | | | | *show (p↔q)
99  | | | | | | | | *show (q→p)
100 | | | | | | | | | p                    81,R
101 | | | | | | | | *show (p→q)
102 | | | | | | | | | q                    92,R
103 | | | | | | | | (p↔q)                  101,99,CB
104 | | | | | (q↔r)                        91,83,CB
105 | | | (p↔(q↔r))                        80,57,CB
106 | | (((p↔q)↔r)↔(p↔(q↔r)))              54,2,CB

CPU time: 276 msec     Statements: 24359
```

```
 1 *show (((p+q)→(p+r))→(p+(q→r)))
 2 | ((p+q)→(p+r))                          ASSUME
 3 | *show (p+(q→r))
 4 | | ¬(p+(q→r))                           ASSUME
 5 | | *show p
 6 | | | ¬p                                ASSUME
 7 | | | *show (q→r)
 8 | | | | q                              ASSUME
 9 | | | | *show r
10 | | | | | ¬r                           ASSUME
11 | | | | | *show (p+q)
12 | | | | | | (p+q)                      8,ADD
13 | | | | | (p+r)                        2,11,MP
14 | | | | | r                            13,6,MTP
15 | | | | ¬(p+(q→r))                       4,R
16 | | | | (p+(q→r))                        7,ADD
17 | | | (p+(q→r))                          5,ADD
```

CPU time: 69 msec
Statements: 3704

```
1 *show ((¬p→q)→(¬q→p))
2 | (¬p→q)                    ASSUME
3 | *show (¬q→p)
4 | | ¬q                     ASSUME
5 | | p                      2,4,MT
```

CPU time: 42 msec
Statements: 919

```
 1 *show (((p&q)+(¬p&q))+((p&¬q)+(¬p&¬q)))
 2 | ¬(((p&q)+(¬p&q))+((p&¬q)+(¬p&¬q)))                    ASSUME
 3 | *show ((p&q)+(¬p&q))
 4 | | ¬((p&q)+(¬p&q))                                     ASSUME
 5 | | *show ((p&¬q)+(¬p&¬q))
 6 | | | ¬((p&¬q)+(¬p&¬q))                                ASSUME
 7 | | | *show (p&q)
 8 | | | | *show p
 9 | | | | ¬p                                           ASSUME
10 | | | | *show (¬p&q)
11 | | | | | ¬p                                         9,R
12 | | | | | *show q
13 | | | | | | ¬q                                       ASSUME
14 | | | | | | *show (p&¬q)
15 | | | | | | | ¬q                                     13,R
16 | | | | | | | *show (¬p&¬q)
17 | | | | | | | | (¬p&¬q)                              11,15,ADJ
18 | | | | | | | ¬((p&¬q)+(¬p&¬q))                      6,R
19 | | | | | | | ((p&¬q)+(¬p&¬q))                       16,ADD
20 | | | | | | ¬((p&¬q)+(¬p&¬q))                        6,R
21 | | | | | | ((p&¬q)+(¬p&¬q))                         14,ADD
22 | | | | | (¬p&q)                                     11,12,ADJ
23 | | | | ¬((p&q)+(¬p&q))                              4,R
24 | | | | ((p&q)+(¬p&q))                               10,ADD
25 | | | *show q
26 | | | | ¬q                                           ASSUME
27 | | | | *show (¬p&q)
28 | | | | | *show ¬p
29 | | | | | | p                                      ASSUME
30 | | | | | | *show (p&¬q)
31 | | | | | | | (p&¬q)                               29,26,ADJ
32 | | | | | | ¬((p&¬q)+(¬p&¬q))                      6,R
33 | | | | | | ((p&¬q)+(¬p&¬q))                       30,ADD
34 | | | | | p                                        8,R
35 | | | | ¬((p&q)+(¬p&q))                              4,R
36 | | | | ((p&q)+(¬p&q))                               27,ADD
37 | | | (p&q)                                          8,25,ADJ
38 | | ¬((p&q)+(¬p&q))                                  4,R
39 | | ((p&q)+(¬p&q))                                   7,ADD
40 | ¬(((p&q)+(¬p&q))+((p&¬q)+(¬p&¬q)))                  2,R
41 | (((p&q)+(¬p&q))+((p&¬q)+(¬p&¬q)))                   5,ADD
42 ((((p&q)+(¬p&q))+((p&¬q)+(¬p&¬q)))                   3,ADD
```

CPU time: 122 msec
Statements: 9463

The conversion to clausal form requires the validity of certain propositional equivalences and certain quantifier equivalences (about moving quantifiers through connectives to the front of a formula). A resolution based prover cannot prove these since it *assumes* them. A partial list is given in Chapter VII, Section C. Some of them are.

12. $\vdash ((p \leftrightarrow q) \leftrightarrow ((q \rightarrow \neg p) \& (\neg q \rightarrow p)))$

13. $\vdash ((p \rightarrow (q \& r)) \leftrightarrow ((p \rightarrow q) \& (p \rightarrow r)))$

14. $\vdash ((Ax)(P^{\circ} \leftrightarrow Qx) \leftrightarrow (P^{\circ} \leftrightarrow (Ax)Qx))$

(The P° in 14 indicates any formula with no free occurrence of x, the variable of quantification.)

```
 1 *show ((p↔q)↔((q+¬p)&(¬q+p)))
 2 | *show (((q+¬p)&(¬q+p))→(p↔q))
 3 | | ((q+¬p)&(¬q+p))                    ASSUME
 4 | | *show (p↔q)
 5 | | | *show (q→p)
 6 | | | | q                            ASSUME
 7 | | | | *show p
 8 | | | | | ¬p                         ASSUME
 9 | | | | | (q+¬p)                     3,S
10 | | | | | (¬q+p)                     3,S
11 | | | | | p                          10,6,MTP
12 | | | *show (p→q)
13 | | | | p                            ASSUME
14 | | | | *show q
15 | | | | | ¬q                         ASSUME
16 | | | | | (q+¬p)                     3,S
17 | | | | | (¬q+p)                     3,S
18 | | | | | ¬p                         15,16,MTP
19 | | | | | p                          13,R
20 | | | (p↔q)                            12,5,CB
21 | | *show ((p↔q)→((q+¬p)&(¬q+p)))
22 | | | (p↔q)                            ASSUME
23 | | | *show ((q+¬p)&(¬q+p))
24 | | | | *show (q+¬p)
25 | | | | | ¬(q+¬p)                    ASSUME
26 | | | | | (p→q)                      22,BC
27 | | | | | (q→p)                      22,BC
28 | | | | | *show q
29 | | | | | | ¬q                       ASSUME
30 | | | | | | ¬p                       29,26,MT
31 | | | | | | (q+¬p)                   30,ADD
32 | | | | | | ¬(q+¬p)                  25,R
33 | | | | | (q+¬p)                     28,ADD
34 | | | | *show (¬q+p)
35 | | | | | ¬(¬q+p)                    ASSUME
36 | | | | | (p→q)                      22,BC
37 | | | | | (q→p)                      22,BC
38 | | | | | *show ¬q
39 | | | | | | q                        ASSUME
40 | | | | | | p                        37,39,MP
41 | | | | | | (¬q+p)                   40,ADD
42 | | | | | | ¬(¬q+p)                  35,R
43 | | | | | (¬q+p)                     38,ADD
44 | | | | ((q+¬p)&(¬q+p))              24,34,ADJ
45 | | ((p↔q)↔((q+¬p)&(¬q+p)))           21,2,CB
```

CPU time: 132 msec
Statements: 9784

```
 1 *show ((p+(q&r))↔((p+q)&(p+r)))
 2 | *show (((p+q)&(p+r))→(p+(q&r)))
 3 | | ((p+q)&(p+r))                          ASSUME
 4 | | *show (p+(q&r))
 5 | | | ¬(p+(q&r))                          ASSUME
 6 | | | (p+q)                               3,S
 7 | | | (p+r)                               3,S
 8 | | | *show p
 9 | | | | ¬p                                ASSUME
10 | | | | q                                 9,6,MTP
11 | | | | r                                 9,7,MTP
12 | | | | *show (q&r)
13 | | | | | (q&r)                           10,11,ADJ
14 | | | | ¬(p+(q&r))                        5,R
15 | | | | (p+(q&r))                         12,ADD
16 | | | (p+(q&r))                           8,ADD
17 | *show ((p+(q&r))→((p+q)&(p+r)))
18 | | (p+(q&r))                             ASSUME
19 | | *show ((p+q)&(p+r))
20 | | | *show (p+q)
21 | | | | ¬(p+q)                            ASSUME
22 | | | | *show p
23 | | | | | ¬p                              ASSUME
24 | | | | | (q&r)                           23,18,MTP
25 | | | | | q                               24,S
26 | | | | | r                               24,S
27 | | | | | (p+q)                           25,ADD
28 | | | | | ¬(p+q)                          21,R
29 | | | | (p+q)                             22,ADD
30 | | | *show (p+r)
31 | | | | ¬(p+r)                            ASSUME
32 | | | | *show p
33 | | | | | ¬p                              ASSUME
34 | | | | | (q&r)                           33,18,MTP
35 | | | | | q                               33,20,MTP
36 | | | | | r                               34,S
37 | | | | | (p+r)                           36,ADD
38 | | | | | ¬(p+r)                          31,R
39 | | | | (p+r)                             32,ADD
40 | | | ((p+q)&(p+r))                       20,30,ADJ
41 | | ((p+(q&r))↔((p+q)&(p+r)))             17,2,CB

CPU time: 121 msec
Statements: 9461
```

```
 1 *show ((Ax1)(P0°+Q1¦(x1))↔(P0°+(Ax1)Q1¦(x1)))
 2 | *show ((P0°+(Ax1)Q1¦(x1))→(Ax1)(P0°+Q1¦(x1)))
 3 | | (P0°+(Ax1)Q1¦(x1))                                   ASSUME
 4 | | *show (Ax1)(P0°+Q1¦(x1))
 5 | | | *show (P0°+Q1¦(x1))
 6 | | | | ¬(P0°+Q1¦(x1))                                   ASSUME
 7 | | | | *show P0°
 8 | | | | | ¬P0°                                         ASSUME
 9 | | | | | (Ax1)Q1¦(x1)                                 8,3,MTP
10 | | | | | Q1¦(x1)                                      9,UI
11 | | | | | (P0°+Q1¦(x1))                                10,ADD
12 | | | | | ¬(P0°+Q1¦(x1))                               6,R
13 | | | | (P0°+Q1¦(x1))                                  7,ADD
14 | *show ((Ax1)(P0°+Q1¦(x1))→(P0°+(Ax1)Q1¦(x1)))
15 | | (Ax1)(P0°+Q1¦(x1))                                   ASSUME
16 | | *show (P0°+(Ax1)Q1¦(x1))
17 | | | ¬(P0°+(Ax1)Q1¦(x1))                              ASSUME
18 | | | (P0°+Q1¦(z2))                                    15,UI
19 | | | *show P0°
20 | | | | ¬P0°                                         ASSUME
21 | | | | Q1¦(z2)                                      20,18,MTP
22 | | | | *show (Ax1)Q1¦(x1)
23 | | | | | *show Q1¦(x1)
24 | | | | | | ¬Q1¦(x1)                                   ASSUME
25 | | | | | | (P0°+Q1¦(x1))                              15,UI
26 | | | | | | Q1¦(x1)                                    25,20,MTP
27 | | | | | ¬(P0°+(Ax1)Q1¦(x1))                          17,R
28 | | | | | (P0°+(Ax1)Q1¦(x1))                           22,ADD
29 | | | | (P0°+(Ax1)Q1¦(x1))                             19,ADD
30 | ((Ax1)(P0°+Q1¦(x1))↔(P0°+(Ax1)Q1¦(x1)))              14,2,CB
```

CPU time: 127 msec
Statements: 8664

In Chapter 7, Section D, (15) was discussed in detail. Other examples here show THINKER proving non-trivial predicate logic arguments.

15. $(Ex)(P^{o} \rightarrow Qx),\ (Ex)(Qx \rightarrow P^{o}) \vdash (Ex)(P^{o} \leftrightarrow Qx)$

16. $\neg(Ex)(Sx \& Qx),\ (Ax)(Px \rightarrow (Qx + Rx)),\ \neg(Ex)Px \rightarrow (Ex)Qx,$
$(Ax)((Qx + Rx) \rightarrow Sx) \vdash (Ex)(Px \& Rx)$

17. $(Ex)Px,\ (Ax)(S_1x \rightarrow (\neg S_2x + \neg Rx)),\ (Ax)(Px \rightarrow (S_1x \& S_2x)),$
$(Ax)(Px \rightarrow Qx) + (Ex)(Px \& Rx) \vdash (Ex)(Qx \& Px)$

18. $(Ex)Px \leftrightarrow (Ex)Qx,\ (Ax)(Ay)((Px \& Qx) \rightarrow (Rx \leftrightarrow Sx))$
$\vdash (Ax)(Px \rightarrow Rx) \leftrightarrow (Ax)(Qx \rightarrow Sx)$

19. $(Ax)(Px \rightarrow Rx),\ (Ax)((S_1x \& S_2x) \rightarrow Px),$
$(Ex)(Rx \& Qx) \rightarrow (Ax)(S_1x \rightarrow \neg Rx) \vdash (Ax)(S_2x \rightarrow \neg S_1x)$

20. $(Ax)P_1x \rightarrow (Ax)Qx,\ (Ax)(Qx + Rx) \rightarrow (Ex)(Qx \& Sx),$
$(Ex)Sx \rightarrow (Ax)(P_2x \rightarrow P_3x) \vdash (Ax)((P_1x \& P_2x) \rightarrow P_3x)$

21. $\vdash [((Ex)Px \leftrightarrow (Ex)Qx) \& (Ax)(Ay)((Px \& Qy) \rightarrow (Sx \leftrightarrow Ry))] \rightarrow$
$((Ax)(Px \rightarrow Sx) \leftrightarrow (Ax)(Qx \rightarrow Rx))$

```
 1 *show (Ex,)(P:↔Q:(x,))
 2 | ¬(Ex,)(P:↔Q:(x,))                    ASSUME
 3 | (Ex,)(P:→Q:(x,))                    PREM
 4 | (P:→Q:(z,))                         3,EI
 5 | (Ex,)(Q:(x,)→P:)                    PREM
 6 | (Q:(z,)→P:)                         5,EI
 7 | (Ax,)¬(P:↔Q:(x,))                   2,QN
 8 | ¬(P:↔Q:(z,))                        7,UI
 9 | ¬(P:↔Q:(z,))                        7,UI
10 | *show (P:↔Q:(z,))
11 | | *show (Q:(z,)→P:)
12 | | | Q:(z,)                          ASSUME
13 | | | *show P:
14 | | | | ¬P:                          ASSUME
15 | | | | ¬Q:(z,)                      14,6,MT
16 | | | | *show (P:↔Q:(z,))
17 | | | | | (Q:(z,)→P:)                6,R
18 | | | | | *show (P:→Q:(z,))
19 | | | | | | P:                       ASSUME
20 | | | | | | ¬P:                      14,R
21 | | | | | (P:↔Q:(z,))                18,17,CB
22 | | | ¬(P:↔Q:(z,))                   9,R
23 | | (P:↔Q:(z,))                      4,11,CB
```

CPU time: 103 msec
Statements: 6651

```
1  *show (Ex1)(P1(x1)&R1(x1))
2  | ¬(Ex1)(P1(x1)&R1(x1))                          ASSUME
3  | ¬(Ex1)(S1(x1)&Q1(x1))                          PREM
4  | (Ax1)¬(S1(x1)&Q1(x1))                          3,QN
5  | (Ax1)¬(P1(x1)&R1(x1))                          2,QN
6  | (Ax1)(P1(x1)→(Q1(x1)+R1(x1)))                  PREM
7  | (P1(z1)→(Q1(z1)+R1(z1)))                       6,UI
8  | (Ax1)((Q1(x1)+R1(x1))→S1(x1))                  PREM
9  | ((Q1(z1)+R1(z1))→S1(z1))                       8,UI
10 | ¬(S1(z1)&Q1(z1))                               4,UI
11 | ¬(P1(z1)&R1(z1))                               5,UI
12 | *show (S1(z1)&Q1(z1))
13 | | *show S1(z1)
14 | | | ¬S1(z1)                                   ASSUME
15 | | | ¬(Q1(z1)+R1(z1))                          14,9,MT
16 | | | ¬P1(z1)                                   15,7,MT
17 | | | *show Q1(z1)
18 | | | | ¬Q1(z1)                                 ASSUME
19 | | | | *show R1(z1)
20 | | | | | ¬R1(z1)                               ASSUME
21 | | | | | *show (P1(z1)&R1(z1))
22 | | | | | | *show P1(z1)
23 | | | | | | | ¬P1(z1)                             ASSUME
24 | | | | | | | *show ¬(Ex1)P1(x1)
25 | | | | | | | | (Ex1)P1(x1)                       ASSUME
26 | | | | | | | | P1(z1)                            25,EI
27 | | | | | | | | (Ax1)(P1(x1)→(Q1(x1)+R1(x1)))     PREM
28 | | | | | | | | (P1(z1)→(Q1(z1)+R1(z1)))          27,UI
29 | | | | | | | | (Ax1)((Q1(x1)+R1(x1))→S1(x1))     PREM
30 | | | | | | | | ((Q1(z1)+R1(z1))→S1(z1))          29,UI
31 | | | | | | | | ¬(S1(z1)&Q1(z1))                  4,UI
32 | | | | | | | | ¬(P1(z1)&R1(z1))                  5,UI
33 | | | | | | | | (Q1(z1)+R1(z1))                   28,26,MP
34 | | | | | | | | S1(z1)                            30,33,MP
35 | | | | | | | | *show (S1(z1)&Q1(z1))
36 | | | | | | | | | S1(z1)                          34,R
37 | | | | | | | | | *show Q1(z1)
38 | | | | | | | | | | ¬Q1(z1)                       ASSUME
39 | | | | | | | | | | R1(z1)                        38,33,MTP
40 | | | | | | | | | | (P1(z1)&R1(z1))               26,39,ADJ
41 | | | | | | | | | | ¬(P1(z1)&R1(z1))              32,R
42 | | | | | | | | | (S1(z1)&Q1(z1))                 36,37,ADJ
43 | | | | | | | (¬(Ex1)P1(x1)→(Ex1)Q1(x1))          PREM
44 | | | | | | | (Ex1)Q1(x1)                         43,24,MP
45 | | | | | | | Q1(z1)                              44,EI
46 | | | | | | | (Ax1)¬P1(x1)                        24,QN
47 | | | | | | | (Ax1)(P1(x1)→(Q1(x1)+R1(x1)))       PREM
```

```
48 | | | | | | | (P¦(z₁)→(Q¦(z₁)+R¦(z₁)))                    47,UI
49 | | | | | | | (Ax₁)((Q¦(x₁)+R¦(x₁))→S¦(x₁))               PREM
50 | | | | | | | ((Q¦(z₁)+R¦(z₁))→S¦(z₁))                    49,UI
51 | | | | | | | ¬(S¦(z₁)&Q¦(z₁))                           4,UI
52 | | | | | | | ¬(P¦(z₁)&R¦(z₁))                           5,UI
53 | | | | | | | ¬P¦(z₁)                                    46,UI
54 | | | | | | | *show (S¦(z₁)&Q¦(z₁))
55 | | | | | | | | *show S¦(z₁)
56 | | | | | | | | | ¬S¦(z₁)                                  ASSUME
57 | | | | | | | | | ¬(Q¦(z₁)+R¦(z₁))                         56,50,MT
58 | | | | | | | | | (Q¦(z₁)+R¦(z₁))                          45,ADD
59 | | | | | | | | (S¦(z₁)&Q¦(z₁))                            55,45,ADJ
60 | | | | | | | ¬P¦(z₁)                                      16,R
61 | | | | | | ¬(P¦(z₁)&R¦(z₁))                               11,R
62 | | | | | ¬(Q¦(z₁)+R¦(z₁))                                 15,R
63 | | | | (Q¦(z₁)+R¦(z₁))                                    19,ADD
64 | | | (Q¦(z₁)+R¦(z₁))                                      17,ADD
65 | | *show Q¦(z₁)
66 | | | ¬Q¦(z₁)                                            ASSUME
67 | | | *show (P¦(z₁)&R¦(z₁))
68 | | | | *show P¦(z₁)
69 | | | | | ¬P¦(z₁)                                        ASSUME
70 | | | | | *show ¬(Ex₁)P¦(x₁)
71 | | | | | | (Ex₁)P¦(x₁)                                  ASSUME
72 | | | | | | P¦(z₂)                                       71,EI
73 | | | | | | (Ax₁)(P¦(x₁)→(Q¦(x₁)+R¦(x₁)))                PREM
74 | | | | | | (P¦(z₂)→(Q¦(z₂)+R¦(z₂)))                     73,UI
75 | | | | | | (Ax₁)((Q¦(x₁)+R¦(x₁))→S¦(x₁))                PREM
76 | | | | | | ((Q¦(z₂)+R¦(z₂))→S¦(z₂))                     75,UI
77 | | | | | | ¬(S¦(z₂)&Q¦(z₂))                             4,UI
78 | | | | | | ¬(P¦(z₂)&R¦(z₂))                             5,UI
79 | | | | | | (Q¦(z₂)+R¦(z₂))                              74,72,MP
80 | | | | | | S¦(z₂)                                       76,79,MP
81 | | | | | | *show (S¦(z₂)&Q¦(z₂))
82 | | | | | | | S¦(z₂)                                     80,R
83 | | | | | | | *show Q¦(z₂)
84 | | | | | | | | ¬Q¦(z₂)                                  ASSUME
85 | | | | | | | | R¦(z₂)                                   84,79,MTP
86 | | | | | | | | (P¦(z₂)&R¦(z₂))                          72,85,ADJ
87 | | | | | | | | ¬(P¦(z₂)&R¦(z₂))                         78,R
88 | | | | | | | (S¦(z₂)&Q¦(z₂))                            82,83,ADJ
89 | | | | | | (¬(Ex₁)P¦(x₁)→(Ex₁)Q¦(x₁))                    PREM
90 | | | | | | (Ex₁)Q¦(x₁)                                  89,70,MP
91 | | | | | | Q¦(z₂)                                       90,EI
92 | | | | | | (Ax₁)¬P¦(x₁)                                 70,QN
93 | | | | | | (Ax₁)(P¦(x₁)→(Q¦(x₁)+R¦(x₁)))                PREM
94 | | | | | | (P¦(z₁)→(Q¦(z₁)+R¦(z₁)))                     93,UI
```

```
95  | | | | | (Ax1)((Q¦(x1)+R¦(x1))→S¦(x1))                PREM
96  | | | | | ((Q¦(z1)+R¦(z1))→S¦(z1))                     95,UI
97  | | | | | ¬(S¦(z1)&Q¦(z1))                             4,UI
98  | | | | | ¬(P¦(z1)&R¦(z1))                             5,UI
99  | | | | | ¬P¦(z1)                                      92,UI
100 | | | | | *show (S¦(z1)&Q¦(z1))
101 | | | | | | *show S¦(z1)
102 | | | | | | | ¬S¦(z1)                                    ASSUME
103 | | | | | | | ¬(Q¦(z1)+R¦(z1))                           102,96,MT
104 | | | | | | | (Q¦(z1)+R¦(z1))                            91,ADD
105 | | | | | | (S¦(z1)&Q¦(z1))                            101,91,ADJ
106 | | | | | *show R¦(z1)
107 | | | | | | ¬R¦(z1)                                    ASSUME
108 | | | | | | (Q¦(z1)+R¦(z1))                            7,68,MP
109 | | | | | | R¦(z1)                                     108,66,MTP
110 | | | | | (P¦(z1)&R¦(z1))                              68,106,ADJ
111 | | | | ¬(P¦(z1)&R¦(z1))                               11,R
112 | | | (S¦(z1)&Q¦(z1))                                  13,65,ADJ
```

CPU time: 639 msec
Statements: 53900

```
1  *show (Ex1)(Q1(x1)&P1(x1))
2  | ¬(Ex1)(Q1(x1)&P1(x1))                                  ASSUME
3  | (Ex1)P1(x1)                                            PREM
4  | P1(z1)                                                 3,EI
5  | (Ax1)¬(Q1(x1)&P1(x1))                                  2,QN
6  | (Ax1)(S1(x1)→(¬S1(x1)→¬R1(x1)))                        PREM
7  | (S1(z1)→(¬S1(z1)→¬R1(z1)))                             6,UI
8  | (Ax1)(P1(x1)→(S1(x1)&S1(x1)))                         PREM
9  | (P1(z1)→(S1(z1)&S1(z1)))                              8,UI
10 | ¬(Q1(z1)&P1(z1))                                       5,UI
11 | (S1(z1)&S1(z1))                                        9,4,MP
12 | S1(z1)                                                 11,S
13 | S1(z1)                                                 11,S
14 | (¬S1(z1)→¬R1(z1))                                      7,12,MP
15 | ¬R1(z1)                                                13,14,MTP
16 | *show (Q1(z1)&P1(z1))
17 | | *show Q1(z1)
18 | | | ¬Q1(z1)                                           ASSUME
19 | | | *show ¬(Ax1)(P1(x1)→Q1(x1))
20 | | | | (Ax1)(P1(x1)→Q1(x1))                            ASSUME
21 | | | | (P1(z1)→Q1(z1))                                 20,UI
22 | | | | Q1(z1)                                          21,4,MP
23 | | | | ¬Q1(z1)                                         18,R
24 | | | ((Ax1)(P1(x1)→Q1(x1))→(Ex1)(P1(x1)&R1(x1)))       PREM
25 | | | (Ex1)(P1(x1)&R1(x1))                              19,24,MTP
26 | | | (P1(z1)&R1(z1))                                   25,EI
27 | | | (Ex1)¬(P1(x1)→Q1(x1))                             19,QN
28 | | | (Ax1)(S1(x1)→(¬S1(x1)→¬R1(x1)))                   PREM
29 | | | (S1(z1)→(¬S1(z1)→¬R1(z1)))                        28,UI
30 | | | (Ax1)(P1(x1)→(S1(x1)&S1(x1)))                     PREM
31 | | | (P1(z1)→(S1(z1)&S1(z1)))                          30,UI
32 | | | ¬(Q1(z1)&P1(z1))                                  5,UI
33 | | | P1(z1)                                            26,S
34 | | | R1(z1)                                            26,S
35 | | | (S1(z1)&S1(z1))                                   31,33,MP
36 | | | ¬(P1(z1)→Q1(z1))                                  27,EI
37 | | | (S1(z1)→(¬S1(z1)→¬R1(z1)))                        28,UI
38 | | | (P1(z1)→(S1(z1)&S1(z1)))                          30,UI
39 | | | ¬(Q1(z1)&P1(z1))                                  5,UI
40 | | | S1(z1)                                            35,S
41 | | | S1(z1)                                            35,S
42 | | | (¬S1(z1)→¬R1(z1))                                 29,40,MP
43 | | | ¬R1(z1)                                           41,42,MTP
44 | | (Q1(z1)&P1(z1))                                     17,4,ADJ
```

CPU time: 185 msec
Statements: 15205

```
1  *show ((Ax1)(P¦(x1)→R¦(x1))↔(Ax1)(Q¦(x1)→S¦(x1)))
2  | *show ((Ax1)(Q¦(x1)→S¦(x1))→(Ax1)(P¦(x1)→R¦(x1)))
3  | | (Ax1)(Q¦(x1)→S¦(x1))                                   ASSUME
4  | | *show (Ax1)(P¦(x1)→R¦(x1))
5  | | | *show (P¦(x1)→R¦(x1))
6  | | | | P¦(x1)                                            ASSUME
7  | | | | *show R¦(x1)
8  | | | | | ¬R¦(x1)                                         ASSUME
9  | | | | | ((Ex1)P¦(x1)↔(Ex1)Q¦(x1))                       PREM
10 | | | | | ((Ex1)P¦(x1)→(Ex1)Q¦(x1))                       9,BC
11 | | | | | ((Ex1)Q¦(x1)→(Ex1)P¦(x1))                       9,BC
12 | | | | | (Ax1)(Ay1)((P¦(x1)&Q¦(y1))→(R¦(x1)↔S¦(y1)))     PREM
13 | | | | | (Ay1)((P¦(x1)&Q¦(y1))→(R¦(x1)↔S¦(y1)))          12,UI
14 | | | | | (Q¦(x1)→S¦(x1))                                 3,UI
15 | | | | | ((P¦(x1)&Q¦(x1))→(R¦(x1)↔S¦(x1)))               13,UI
16 | | | | | *show (Ex1)P¦(x1)
17 | | | | | | (Ex1)P¦(x1)                                   6,EG
18 | | | | | (Ex1)Q¦(x1)                                     10,16,MP
19 | | | | | P¦(z2)                                          16,EI
20 | | | | | Q¦(z2)                                          18,EI
21 | | | | | (Ay1)((P¦(z2)&Q¦(y1))→(R¦(z2)↔S¦(y1)))          12,UI
22 | | | | | (Ay1)((P¦(z2)&Q¦(y1))→(R¦(z2)↔S¦(y1)))          12,UI
23 | | | | | (Q¦(z2)→S¦(z2))                                 3,UI
24 | | | | | (Q¦(z2)→S¦(z2))                                 3,UI
25 | | | | | ((P¦(x1)&Q¦(z2))→(R¦(x1)↔S¦(z2)))               13,UI
26 | | | | | ((P¦(x1)&Q¦(z2))→(R¦(x1)↔S¦(z2)))               13,UI
27 | | | | | ((P¦(z2)&Q¦(x1))→(R¦(z2)↔S¦(x1)))               21,UI
28 | | | | | ((P¦(z2)&Q¦(z2))→(R¦(z2)↔S¦(z2)))               21,UI
29 | | | | | ((P¦(z2)&Q¦(z2))→(R¦(z2)↔S¦(z2)))               21,UI
30 | | | | | ((P¦(z2)&Q¦(x1))→(R¦(z2)↔S¦(x1)))               22,UI
31 | | | | | ((P¦(z2)&Q¦(z2))→(R¦(z2)↔S¦(z2)))               22,UI
32 | | | | | ((P¦(z2)&Q¦(z2))→(R¦(z2)↔S¦(z2)))               22,UI
33 | | | | | S¦(z2)                                          24,20,MP
34 | | | | | *show Q¦(x1)
35 | | | | | | ¬Q¦(x1)                                       ASSUME
36 | | | | | | *show (P¦(x1)&Q¦(x1))
37 | | | | | | | P¦(x1)                                      6,R
38 | | | | | | | *show Q¦(z2)
39 | | | | | | | | ¬Q¦(z2)                                   ASSUME
40 | | | | | | | | *show (P¦(x1)&Q¦(z2))
41 | | | | | | | | | P¦(x1)                                  37,R
42 | | | | | | | | | *show (P¦(x1)&Q¦(z2))
43 | | | | | | | | | | (P¦(x1)&Q¦(z2))                       41,20,ADJ
44 | | | | | | | | | (R¦(x1)↔S¦(z2))                         26,42,MP
45 | | | | | | | | | (R¦(x1)→S¦(z2))                         44,BC
46 | | | | | | | | | (S¦(z2)→R¦(x1))                         44,BC
47 | | | | | | | | | R¦(x1)                                  46,33,MP
```

```
48 | | | | | | | | | ¬R¦(x,)                                                    8,R
49 | | | | | | | | | Q¦(z,)                                                     40,S
50 | | | | | | | | S¦(z,)                                                       23,38,MP
51 | | | | | | | | *show (P¦(x,)&Q¦(z,))
52 | | | | | | | | | (P¦(x,)&Q¦(z,))                                            37,38,ADJ
53 | | | | | | | | (R¦(x,)↔S¦(z,))                                            25,51,MP
54 | | | | | | | | (R¦(x,)→S¦(z,))                                            53,BC
55 | | | | | | | | (S¦(z,)→R¦(x,))                                            53,BC
56 | | | | | | | | R¦(x,)                                                       55,50,MP
57 | | | | | | | | ¬R¦(x,)                                                    8,R
58 | | | | | | | | Q¦(x,)                                                     36,S
59 | | | | | | | S¦(x,)                                                       14,34,MP
60 | | | | | | | *show (P¦(x,)&Q¦(x,))
61 | | | | | | | | (P¦(x,)&Q¦(x,))                                            6,34,ADJ
62 | | | | | | | (R¦(x,)↔S¦(x,))                                            15,60,MP
63 | | | | | | | R¦(x,)                                                       62,BC,59,MP
64 | | | | | *show ((Ax,)(P¦(x,)→R¦(x,))→(Ax,)(Q¦(x,)→S¦(x,)))
65 | | | | | | (Ax,)(P¦(x,)→R¦(x,))                                         ASSUME
66 | | | | | | *show (Ax,)(Q¦(x,)→S¦(x,))
67 | | | | | | | *show (Q¦(x,)→S¦(x,))
68 | | | | | | | Q¦(x,)                                                     ASSUME
69 | | | | | | | *show S¦(x,)
70 | | | | | | | | ¬S¦(x,)                                                  ASSUME
71 | | | | | | | | ((Ex,)P¦(x,)↔(Ex,)Q¦(x,))                                PREM
72 | | | | | | | | ((Ex,)P¦(x,)→(Ex,)Q¦(x,))                                71,BC
73 | | | | | | | | ((Ex,)Q¦(x,)→(Ex,)P¦(x,))                                71,BC
74 | | | | | | | | (Ax,)(Ay,)((P¦(x,)&Q¦(y,))→(R¦(x,)↔S¦(y,)))              PREM
75 | | | | | | | | (Ay,)((P¦(x,)&Q¦(y,))→(R¦(x,)↔S¦(y,)))                   74,UI
76 | | | | | | | | (P¦(x,)→R¦(x,))                                          65,UI
77 | | | | | | | | ((P¦(x,)&Q¦(x,))→(R¦(x,)↔S¦(x,)))                        75,UI
78 | | | | | | | | *show (Ex,)P¦(x,)
79 | | | | | | | | | ¬(Ex,)P¦(x,)                                           ASSUME
80 | | | | | | | | | ¬(Ex,)Q¦(x,)                                           79,73,MT
81 | | | | | | | | | (Ax,)¬P¦(x,)                                           79,QN
82 | | | | | | | | | (Ax,)¬Q¦(x,)                                           80,QN
83 | | | | | | | | | ¬P¦(x,)                                                81,UI
84 | | | | | | | | | ¬Q¦(x,)                                                82,UI
85 | | | | | | | | | Q¦(x,)                                                 68,R
86 | | | | | | | | (Ex,)Q¦(x,)                                              72,78,MP
87 | | | | | | | | P¦(z,)                                                   78,EI
88 | | | | | | | | Q¦(z,)                                                   86,EI
89 | | | | | | | | (Ay,)((P¦(z,)&Q¦(y,))→(R¦(z,)↔S¦(y,)))                   74,UI
90 | | | | | | | | (Ay,)((P¦(z,)&Q¦(y,))→(R¦(z,)↔S¦(y,)))                   74,UI
91 | | | | | | | | (P¦(z,)→R¦(z,))                                          65,UI
92 | | | | | | | | (P¦(z,)→R¦(z,))                                          65,UI
93 | | | | | | | | ((P¦(x,)&Q¦(z,))→(R¦(x,)↔S¦(z,)))                        75,UI
94 | | | | | | | | ((P¦(x,)&Q¦(z,))→(R¦(x,)↔S¦(z,)))                        75,UI
```

```
95  | | | | | ((P¦(z₁)&Q¦(x₁))→(R¦(z₁)↔S¦(x₁)))                    89,UI
96  | | | | | ((P¦(z₁)&Q¦(z₁))→(R¦(z₁)↔S¦(z₁)))                    89,UI
97  | | | | | ((P¦(z₁)&Q¦(z₂))→(R¦(z₁)↔S¦(z₂)))                    89,UI
98  | | | | | ((P¦(z₂)&Q¦(x₁))→(R¦(z₂)↔S¦(x₁)))                    90,UI
99  | | | | | ((P¦(z₂)&Q¦(z₁))→(R¦(z₂)↔S¦(z₁)))                    90,UI
100 | | | | | ((P¦(z₂)&Q¦(z₂))→(R¦(z₂)↔S¦(z₂)))                    90,UI
101 | | | | | R¦(z₁)                                              91,87,MP
102 | | | | | *show P¦(x₁)
103 | | | | |  ¬P¦(x₁)                                           ASSUME
104 | | | | |  *show (P¦(x₁)&Q¦(x₁))
105 | | | | |  | Q¦(x₁)                                          68,R
106 | | | | |  | *show P¦(z₂)
107 | | | | |  | | ¬P¦(z₂)                                       ASSUME
108 | | | | |  | | *show (P¦(x₁)&Q¦(z₁))
109 | | | | |  | | | *show Q¦(z₁)
110 | | | | |  | | | | ¬Q¦(z₁)                                   ASSUME
111 | | | | |  | | | | *show (P¦(x₁)&Q¦(z₂))
112 | | | | |  | | | | | Q¦(z₂)                                  88,R
113 | | | | |  | | | | | *show (P¦(z₁)&Q¦(x₁))
114 | | | | |  | | | | | | (P¦(z₁)&Q¦(x₁))                       87,105,ADJ
115 | | | | |  | | | | | (R¦(z₁)↔S¦(x₁))                         95,113,MP
116 | | | | |  | | | | | (R¦(z₁)→S¦(x₁))                         115,BC
117 | | | | |  | | | | | (S¦(x₁)→R¦(z₁))                         115,BC
118 | | | | |  | | | | | S¦(x₁)                                  116,101,MP
119 | | | | |  | | | | | ¬S¦(x₁)                                 70,R
120 | | | | |  | | | | P¦(x₁)                                    111,S
121 | | | | |  | | | | ¬P¦(x₁)                                   103,R
122 | | | | |  | | | *show (P¦(x₁)&Q¦(z₂))
123 | | | | |  | | | | Q¦(z₂)                                    88,R
124 | | | | |  | | | | *show (P¦(z₁)&Q¦(x₁))
125 | | | | |  | | | | | (P¦(z₁)&Q¦(x₁))                         87,105,ADJ
126 | | | | |  | | | | (R¦(z₁)↔S¦(x₁))                           95,124,MP
127 | | | | |  | | | | (R¦(z₁)→S¦(x₁))                           126,BC
128 | | | | |  | | | | (S¦(x₁)→R¦(z₁))                           126,BC
129 | | | | |  | | | | S¦(x₁)                                    127,101,MP
130 | | | | |  | | | | ¬S¦(x₁)                                   70,R
131 | | | | |  | | | P¦(x₁)                                      122,S
132 | | | | |  | | | ¬P¦(x₁)                                     103,R
133 | | | | |  | | P¦(x₁)                                        108,S
134 | | | | |  | | ¬P¦(x₁)                                       103,R
135 | | | | |  | R¦(z₂)                                          92,106,MP
136 | | | | |  | *show (P¦(x₁)&Q¦(z₁))
137 | | | | |  | | *show Q¦(z₁)
138 | | | | |  | | | ¬Q¦(z₁)                                     ASSUME
139 | | | | |  | | | *show (P¦(x₁)&Q¦(z₂))
140 | | | | |  | | | | Q¦(z₂)                                    88,R
141 | | | | |  | | | | *show (P¦(z₁)&Q¦(x₁))
```

```
142 | | | | | | | | | | | (P¦(z,)&Q¦(x,))                                  87,105,ADJ
143 | | | | | | | | | | | (R¦(z,)↔S¦(x,))                                  95,141,MP
144 | | | | | | | | | | | (R¦(z,)→S¦(x,))                                  143,BC
145 | | | | | | | | | | | (S¦(x,)→R¦(z,))                                  143,BC
146 | | | | | | | | | | | S¦(x,)                                           144,101,MP
147 | | | | | | | | | | | ¬S¦(x,)                                          70,R
148 | | | | | | | | | | P¦(x,)                                           139,S
149 | | | | | | | | | | ¬P¦(x,)                                          103,R
150 | | | | | | | | | *show (P¦(x,)&Q¦(z,))
151 | | | | | | | | | | Q¦(z,)                                           88,R
152 | | | | | | | | | | *show (P¦(z,)&Q¦(x,))
153 | | | | | | | | | | | (P¦(z,)&Q¦(x,))                                  87,105,ADJ
154 | | | | | | | | | | | (R¦(z,)↔S¦(x,))                                  95,152,MP
155 | | | | | | | | | | | (R¦(z,)→S¦(x,))                                  154,BC
156 | | | | | | | | | | | (S¦(x,)→R¦(z,))                                  154,BC
157 | | | | | | | | | | | S¦(x,)                                           155,101,MP
158 | | | | | | | | | | | ¬S¦(x,)                                          70,R
159 | | | | | | | | | | P¦(x,)                                           150,S
160 | | | | | | | | | | ¬P¦(x,)                                          103,R
161 | | | | | | | | | P¦(x,)                                           136,S
162 | | | | | | | | | ¬P¦(x,)                                          103,R
163 | | | | | | | | P¦(x,)                                           104,S
164 | | | | | | | R¦(x,)                                           76,102,MP
165 | | | | | | | *show (P¦(x,)&Q¦(x,))
166 | | | | | | | | (P¦(x,)&Q¦(x,))                                  102,68,ADJ
167 | | | | | | | (R¦(x,)↔S¦(x,))                                  77,165,MP
168 | | | | | | | S¦(x,)                                           167,BC,164,MP
169 | | | | | | ((Ax,)(P¦(x,)→R¦(x,))↔(Ax,)(Q¦(x,)→S¦(x,)))       64,2,CB
```

CPU time: 1070 msec
Statements: 85757

```
1  *show (Ax1)(S1(x1)→¬S1(x1))                                                    ASSUME
2  | *show (S1(x1)→¬S1(x1))                                                      ASSUME
3  | | S1(x1)                                                                    PREM
4  | | *show ¬S1(x1)                                                           6,EI
5  | | | S1(x1)                                                                  PREM
6  | | | (Ex1)(P1(x1)&Q1(x1))                                                    8,UI
7  | | | (P1(z1)&Q1(z1))                                                         8,UI
8  | | | (Ax1)(P1(x1)→R1(x1))                                                    PREM
9  | | | (P1(x1)→R1(x1))                                                         11,UI
10 | | | (P1(z1)→R1(z1))                                                         11,UI
11 | | | (Ax1)((S1(x1)&S1(x1))→P1(x1))                                           PREM
12 | | | ((S1(x1)&S1(x1))→P1(x1))                                                11,UI
13 | | | ((S1(z1)&S1(z1))→P1(z1))                                                11,UI
14 | | | P1(z1)                                                                  7,S
15 | | | Q1(z1)                                                                  7,S
16 | | | R1(z1)                                                                  10,14,MP
17 | | | *show (Ex1)(R1(x1)&Q1(x1))                                              ASSUME
18 | | | | ¬(Ex1)(R1(x1)&Q1(x1))                                                 18,QN
19 | | | | (Ax1)¬(R1(x1)&Q1(x1))                                                 19,UI
20 | | | | ¬(R1(x1)&Q1(x1))                                                      19,UI
21 | | | | ¬(R1(z1)&Q1(z1))                                                      21,R
22 | | | | *show (R1(x1)&Q1(x1))                                                 ASSUME
23 | | | | | *show R1(x1)                                                        24,9,MT
24 | | | | | | ¬R1(x1)                                                           25,12,MT
25 | | | | | | ¬P1(x1)                                                           16,15,ADJ
26 | | | | | | ¬(S1(x1)&S1(x1))                                                  21,R
27 | | | | | | (R1(z1)&Q1(z1))                                                   ASSUME
28 | | | | | | ¬(R1(z1)&Q1(z1))                                                  16,15,ADJ
29 | | | | | *show Q1(x1)                                                        21,R
30 | | | | | | ¬Q1(x1)                                                           23,29,ADJ
31 | | | | | | (R1(z1)&Q1(z1))                                                   PREM
32 | | | | | | ¬(R1(z1)&Q1(z1))                                                  34,17,MP
33 | | | | | (R1(x1)&Q1(x1))                                                     17,EI
34 | | | ((Ex1)(R1(x1)&Q1(x1))→(Ax1)(S1(x1)→¬R1(x1)))                            8,UI
35 | | | (Ax1)(S1(x1)→¬R1(x1))                                                   11,UI
36 | | | (R1(z1)&Q1(z1))                                                         35,UI
37 | | | (P1(z1)→R1(z1))                                                         35,UI
38 | | | ((S1(z1)&S1(z1))→P1(z1))                                                35,UI
39 | | | (S1(x1)→¬R1(x1))                                                        36,S
40 | | | (S1(z1)→¬R1(z1))                                                        36,S
41 | | | (S1(z1)→¬R1(z1))                                                        39,5,MP
42 | | | R1(z1)                                                                  44,9,MT
43 | | | Q1(z1)                                                                  45,12,MT
44 | | | ¬R1(x1)                                                                 3,5,ADJ
45 | | | ¬P1(x1)
46 | | | ¬(S1(x1)&S1(x1))
47 | | | (S1(x1)&S1(x1))
```

CPU time: 219 msec
Statements: 18573

```
1  *show (Ax1)((P1(x1)&P1(x1))→P1(x1))
2  | *show ((P1(x1)&P1(x1))→P1(x1))
3  | | (P1(x1)&P1(x1))                                                    ASSUME
4  | | *show P1(x1)
5  | | | ¬P1(x1)                                                         ASSUME
6  | | | P1(x1)                                                          3,S
7  | | | P1(x1)                                                          3,S
8  | | | (Ax1)(P1(x1)→(Ax1)Q1(x1))                                       PREM
9  | | | (P1(x1)→(Ax1)Q1(x1))                                            8,UI
10 | | | (Ax1)Q1(x1)                                                     9,6,MP
11 | | | Q1(x1)                                                          10,UI
12 | | | *show (Ax1)(Q1(x1)→R1(x1))
13 | | | | ¬(Ax1)(Q1(x1)→R1(x1))                                         ASSUME
14 | | | | (Ex1)¬(Q1(x1)→R1(x1))                                         13,QN
15 | | | | ¬(Q1(z1)→R1(z1))                                              14,EI
16 | | | | *show Q1(z1)
17 | | | | | Q1(z1)                                                      10,UI
18 | | | | (Q1(z1)→R1(z1))                                               16,ADD
19 | | | ((Ax1)(Q1(x1)→R1(x1))→(Ex1)(Q1(x1)&S1(x1)))                     PREM
20 | | | (Ex1)(Q1(x1)&S1(x1))                                            19,12,MP
21 | | | (Q1(z1)&S1(z1))                                                 20,EI
22 | | | (Q1(x1)→R1(x1))                                                 12,UI
23 | | | Q1(z1)                                                          21,S
24 | | | S1(z1)                                                          21,S
25 | | | *show (Ex1)S1(x1)
26 | | | | (Ex1)S1(x1)                                                   24,EG
27 | | | ((Ex1)S1(x1)→(Ax1)(P1(x1)→P1(x1)))                              PREM
28 | | | (Ax1)(P1(x1)→P1(x1))                                            27,25,MP
29 | | | S1(z1)                                                          25,EI
30 | | | (P1(z1)→(Ax1)Q1(x1))                                            8,UI
31 | | | Q1(z1)                                                          10,UI
32 | | | (Q1(z1)→R1(z1))                                                 12,UI
33 | | | (P1(x1)→P1(x1))                                                 28,UI
34 | | | P1(x1)                                                          33,7,MP
```

CPU time: 147 msec
Statements: 11443

```
1  *show ((((Ex1)P1(x1)↔(Ex1)Q1(x1))&(Ax1)(Ay1)((P1(x1)&Q1(y1))→(S1(x1)↔R1(y1))))→((Ax1)(P1(x1)→S1(x1))↔(Ax1)(Q1(x1)→R1(x1))))
2  | (((Ex1)P1(x1)↔(Ex1)Q1(x1))&(Ax1)(Ay1)((P1(x1)&Q1(y1))→(S1(x1)↔R1(y1))))        ASSUME
3  | *show ((Ax1)(P1(x1)→S1(x1))↔(Ax1)(Q1(x1)→R1(x1)))
4  | | *show ((Ax1)(Q1(x1)→R1(x1))→(Ax1)(P1(x1)→S1(x1)))
5  | | | (Ax1)(Q1(x1)→R1(x1))                                                      ASSUME
6  | | | *show (Ax1)(P1(x1)→S1(x1))
7  | | | | *show (P1(x1)→S1(x1))
8  | | | | | P1(x1)                                                               ASSUME
9  | | | | | *show S1(x1)
10 | | | | | | ¬S1(x1)                                                            ASSUME
11 | | | | | | ((Ex1)P1(x1)↔(Ex1)Q1(x1))                                          2,S
12 | | | | | | (Ax1)(Ay1)((P1(x1)&Q1(y1))→(S1(x1)↔R1(y1)))                        2,S
13 | | | | | | (Q1(x1)→R1(x1))                                                    5,UI
14 | | | | | | (Ay1)((P1(x1)&Q1(y1))→(S1(x1)↔R1(y1)))                             12,UI
15 | | | | | | ((P1(x1)&Q1(x1))→(S1(x1)↔R1(x1)))                                  14,UI
16 | | | | | | ((Ex1)P1(x1)→(Ex1)Q1(x1))                                          11,BC
17 | | | | | | ((Ex1)Q1(x1)→(Ex1)P1(x1))                                          11,BC
18 | | | | | | *show Q1(x1)
19 | | | | | | | ¬Q1(x1)                                                          ASSUME
20 | | | | | | | *show (P1(x1)&Q1(x1))
21 | | | | | | | | P1(x1)                                                         8,R
22 | | | | | | | | *show (Ex1)P1(x1)
23 | | | | | | | | | (Ex1)P1(x1)                                                  21,EG
24 | | | | | | | | (Ex1)Q1(x1)                                                    16,22,MP
25 | | | | | | | | P1(z1)                                                         22,EI
26 | | | | | | | | Q1(z1)                                                         24,EI
27 | | | | | | | | (Q1(z1)→R1(z1))                                                5,UI
28 | | | | | | | | (Q1(z1)→R1(z1))                                                5,UI
29 | | | | | | | | (Ay1)((P1(z1)&Q1(y1))→(S1(z1)↔R1(y1)))                         12,UI
30 | | | | | | | | (Ay1)((P1(z1)&Q1(y1))→(S1(z1)↔R1(y1)))                         12,UI
31 | | | | | | | | ((P1(x1)&Q1(z1))→(S1(x1)↔R1(z1)))                              14,UI
32 | | | | | | | | ((P1(x1)&Q1(z1))→(S1(x1)↔R1(z1)))                              14,UI
33 | | | | | | | | ((P1(z1)&Q1(x1))→(S1(z1)↔R1(x1)))                              29,UI
34 | | | | | | | | ((P1(z1)&Q1(z1))→(S1(z1)↔R1(z1)))                              29,UI
35 | | | | | | | | ((P1(z1)&Q1(z1))→(S1(z1)↔R1(z1)))                              29,UI
36 | | | | | | | | ((P1(z1)&Q1(x1))→(S1(z1)↔R1(x1)))                              30,UI
37 | | | | | | | | ((P1(z1)&Q1(z1))→(S1(z1)↔R1(z1)))                              30,UI
38 | | | | | | | | ((P1(z1)&Q1(z1))→(S1(z1)↔R1(z1)))                              30,UI
39 | | | | | | | | R1(z1)                                                         28,26,MP
40 | | | | | | | | *show Q1(z1)
41 | | | | | | | | | ¬Q1(z1)                                                      ASSUME
42 | | | | | | | | | *show (P1(x1)&Q1(z1))
43 | | | | | | | | | | P1(x1)                                                     21,R
44 | | | | | | | | | | *show (P1(x1)&Q1(z1))
45 | | | | | | | | | | | (P1(x1)&Q1(z1))                                          43,26,ADJ
46 | | | | | | | | | | (S1(x1)↔R1(z1))                                            32,44,MP
47 | | | | | | | | | | (S1(x1)→R1(z1))                                            46,BC
```

```
48 | | | | | | | | | | (R¦(z₂)→S¦(x₁))                                                    46,BC
49 | | | | | | | | | | S¦(x₁)                                                             48,39,MP
50 | | | | | | | | | | ¬S¦(x₁)                                                            10,R
51 | | | | | | | | | Q¦(z₂)                                                               42,S
52 | | | | | | | | R¦(z₂)                                                               27,40,MP
53 | | | | | | | | *show (P¦(x₁)&Q¦(z₂))
54 | | | | | | | | | (P¦(x₁)&Q¦(z₂))                                                    21,40,ADJ
55 | | | | | | | | (S¦(x₁)↔R¦(z₂))                                                    31,53,MP
56 | | | | | | | | (S¦(x₁)→R¦(z₂))                                                    55,BC
57 | | | | | | | | (R¦(z₂)→S¦(x₁))                                                    55,BC
58 | | | | | | | | S¦(x₁)                                                             57,52,MP
59 | | | | | | | | ¬S¦(x₁)                                                            10,R
60 | | | | | | | Q¦(x₁)                                                               20,S
61 | | | | | | R¦(x₁)                                                               13,18,MP
62 | | | | | | *show (P¦(x₁)&Q¦(x₁))
63 | | | | | | | (P¦(x₁)&Q¦(x₁))                                                    8,18,ADJ
64 | | | | | | (S¦(x₁)↔R¦(x₁))                                                    15,62,MP
65 | | | | | | S¦(x₁)                                                             64,BC,61,MP
66 | | | | *show ((Ax₁)(P¦(x₁)→S¦(x₁))→(Ax₁)(Q¦(x₁)→R¦(x₁)))
67 | | | | | (Ax₁)(P¦(x₁)→S¦(x₁))                                                 ASSUME
68 | | | | | *show (Ax₁)(Q¦(x₁)→R¦(x₁))
69 | | | | | | *show (Q¦(x₁)→R¦(x₁))                                              ASSUME
70 | | | | | | | Q¦(x₁)                                                             ASSUME
71 | | | | | | | *show R¦(x₁)
72 | | | | | | | | ¬R¦(x₁)                                                          ASSUME
73 | | | | | | | | ((Ex₁)P¦(x₁)↔(Ex₁)Q¦(x₁))                                        2,S
74 | | | | | | | | (Ax₁)(Ay₁)((P¦(x₁)&Q¦(y₁))→(S¦(x₁)↔R¦(y₁)))                      2,S
75 | | | | | | | | (P¦(x₁)→S¦(x₁))                                                  67,UI
76 | | | | | | | | (Ay₁)((P¦(x₁)&Q¦(y₁))→(S¦(x₁)↔R¦(y₁)))                           74,UI
77 | | | | | | | | ((P¦(x₁)&Q¦(x₁))→(S¦(x₁)↔R¦(x₁)))                                76,UI
78 | | | | | | | | ((Ex₁)P¦(x₁)→(Ex₁)Q¦(x₁))                                        73,BC
79 | | | | | | | | ((Ex₁)Q¦(x₁)→(Ex₁)P¦(x₁))                                        73,BC
80 | | | | | | | | *show P¦(x₁)
81 | | | | | | | | | ¬P¦(x₁)                                                        ASSUME
82 | | | | | | | | | *show (P¦(x₁)&Q¦(x₁))
83 | | | | | | | | | | Q¦(x₁)                                                       70,R
84 | | | | | | | | | | *show (Ex₁)P¦(x₁)
85 | | | | | | | | | | | ¬(Ex₁)P¦(x₁)                                               ASSUME
86 | | | | | | | | | | | ¬(Ex₁)Q¦(x₁)                                               85,79,MT
87 | | | | | | | | | | | (Ax₁)¬P¦(x₁)                                               85,QN
88 | | | | | | | | | | | (Ax₁)¬Q¦(x₁)                                               86,QN
89 | | | | | | | | | | | ¬Q¦(x₁)                                                    88,UI
90 | | | | | | | | | | | Q¦(x₁)                                                     83,R
91 | | | | | | | | | | (Ex₁)Q¦(x₁)                                                  78,84,MP
92 | | | | | | | | | | P¦(z₁)                                                       84,EI
93 | | | | | | | | | | Q¦(z₂)                                                       91,EI
94 | | | | | | | | | | (P¦(z₁)→S¦(z₁))                                              67,UI
```

```
95  | | | | | | | | (P¦(z₂)→S¦(z₂))                                                    67,UI
96  | | | | | | | | (Ay₁)((P¦(z₁)&Q¦(y₁))→(S¦(z₁)↔R¦(y₁)))                              74,UI
97  | | | | | | | | (Ay₁)((P¦(z₂)&Q¦(y₁))→(S¦(z₂)↔R¦(y₁)))                              74,UI
98  | | | | | | | | ((P¦(x₁)&Q¦(z₁))→(S¦(x₁)↔R¦(z₁)))                                  76,UI
99  | | | | | | | | ((P¦(x₁)&Q¦(z₂))→(S¦(x₁)↔R¦(z₂)))                                  76,UI
100 | | | | | | | | ((P¦(z₁)&Q¦(x₁))→(S¦(z₁)↔R¦(x₁)))                                  96,UI
101 | | | | | | | | ((P¦(z₁)&Q¦(z₁))→(S¦(z₁)↔R¦(z₁)))                                  96,UI
102 | | | | | | | | ((P¦(z₁)&Q¦(z₂))→(S¦(z₁)↔R¦(z₂)))                                  96,UI
103 | | | | | | | | ((P¦(z₂)&Q¦(x₁))→(S¦(z₂)↔R¦(x₁)))                                  97,UI
104 | | | | | | | | ((P¦(z₂)&Q¦(z₁))→(S¦(z₂)↔R¦(z₁)))                                  97,UI
105 | | | | | | | | ((P¦(z₂)&Q¦(z₂))→(S¦(z₂)↔R¦(z₂)))                                  97,UI
106 | | | | | | | | S¦(z₁)                                                            94,92,MP
107 | | | | | | | | *show P¦(z₂)
108 | | | | | | | |  ¬P¦(z₂)                                                          ASSUME
109 | | | | | | | |  *show (P¦(x₁)&Q¦(z₁))
110 | | | | | | | |  | *show Q¦(z₁)
111 | | | | | | | |  | | ¬Q¦(z₁)                                                       ASSUME
112 | | | | | | | |  | | *show (P¦(x₁)&Q¦(z₂))
113 | | | | | | | |  | | | Q¦(z₂)                                                      93,R
114 | | | | | | | |  | | | *show (P¦(z₁)&Q¦(x₁))
115 | | | | | | | |  | | | | (P¦(z₁)&Q¦(x₁))                                           92,83,ADJ
116 | | | | | | | |  | | | (S¦(z₁)↔R¦(x₁))                                             100,114,MP
117 | | | | | | | |  | | | (S¦(z₁)→R¦(x₁))                                             116,BC
118 | | | | | | | |  | | | (R¦(x₁)→S¦(z₁))                                             116,BC
119 | | | | | | | |  | | | R¦(x₁)                                                      117,106,MP
120 | | | | | | | |  | | | ¬R¦(x₁)                                                     72,R
121 | | | | | | | |  | | P¦(x₁)                                                        112,S
122 | | | | | | | |  | | ¬P¦(x₁)                                                       81,R
123 | | | | | | | |  | *show (P¦(x₁)&Q¦(z₂))
124 | | | | | | | |  | | Q¦(z₂)                                                      93,R
125 | | | | | | | |  | | *show (P¦(z₁)&Q¦(x₁))
126 | | | | | | | |  | | | (P¦(z₁)&Q¦(x₁))                                           92,83,ADJ
127 | | | | | | | |  | | (S¦(z₁)↔R¦(x₁))                                             100,125,MP
128 | | | | | | | |  | | (S¦(z₁)→R¦(x₁))                                             127,BC
129 | | | | | | | |  | | (R¦(x₁)→S¦(z₁))                                             127,BC
130 | | | | | | | |  | | R¦(x₁)                                                      128,106,MP
131 | | | | | | | |  | | ¬R¦(x₁)                                                     72,R
132 | | | | | | | |  | P¦(x₁)                                                        123,S
133 | | | | | | | |  | ¬P¦(x₁)                                                       81,R
134 | | | | | | | |  P¦(x₁)                                                         109,S
135 | | | | | | | |  ¬P¦(x₁)                                                        81,R
136 | | | | | | | | S¦(z₂)                                                            95,107,MP
137 | | | | | | | | *show (P¦(x₁)&Q¦(z₁))
138 | | | | | | | |  | *show Q¦(z₁)
139 | | | | | | | |  | | ¬Q¦(z₁)                                                       ASSUME
140 | | | | | | | |  | | *show (P¦(x₁)&Q¦(z₂))
141 | | | | | | | |  | | | Q¦(z₂)                                                      93,R
```

```
142 | | | | | | | | | | | *show (P¦(z,)&Q¦(x,))                                  92,83,ADJ
143 | | | | | | | | | | |  | (P¦(z,)&Q¦(x,))                                     100,142,MP
144 | | | | | | | | | | |  (S¦(z,)↔R¦(x,))                                     144,BC
145 | | | | | | | | | | |  (S¦(z,)→R¦(x,))                                     144,BC
146 | | | | | | | | | | |  (R¦(x,)→S¦(z,))                                     145,106,MP
147 | | | | | | | | | | |  R¦(x,)                                              72,R
148 | | | | | | | | | | |  ¬R¦(x,)                                             140,S
149 | | | | | | | | | |  P¦(x,)                                              81,R
150 | | | | | | | | | |  ¬P¦(x,)
151 | | | | | | | | | *show (P¦(x,)&Q¦(z,))                                  93,R
152 | | | | | | | | |  Q¦(z,)
153 | | | | | | | | |  *show (P¦(z,)&Q¦(x,))                                 92,83,ADJ
154 | | | | | | | | |  | (P¦(z,)&Q¦(x,))                                     100,153,MP
155 | | | | | | | | |  (S¦(z,)↔R¦(x,))                                     155,BC
156 | | | | | | | | |  (S¦(z,)→R¦(x,))                                     155,BC
157 | | | | | | | | |  (R¦(x,)→S¦(z,))                                     156,106,MP
158 | | | | | | | | |  R¦(x,)                                              72,R
159 | | | | | | | | |  ¬R¦(x,)                                             151,S
160 | | | | | | | |  P¦(x,)                                              81,R
161 | | | | | | | |  ¬P¦(x,)                                             137,S
162 | | | | | | |  P¦(x,)                                              81,R
163 | | | | | | |  ¬P¦(x,)                                             82,S
164 | | | | | |  P¦(x,)                                              75,80,MP
165 | | | | |  S¦(x,)
166 | | | | |  *show (P¦(x,)&Q¦(x,))                                 80,70,ADJ
167 | | | | |  | (P¦(x,)&Q¦(x,))                                     77,166,MP
168 | | | | |  (S¦(x,)↔R¦(x,))                                     168,BC,165,MP
169 | | | | |  R¦(x,)                                              66,4,CB
170 | | ((Ax,)(P¦(x,)→S¦(x,))↔(Ax,)(Q¦(x,)→R¦(x,)))
```

CPU time: 1076 msec
Statements: 86637

The set theoretic examples given to THINKER were discussed in Chapter VII, Section I. Letting P_1^2 stand for "is a member of" THINKER proves such things as: there is no "Russell set", that if there were an "anti Russell set" then not every set has a complement, that given the axiom of separation there is no "universal set", and that there is no set of "non-circular sets". (25) is the proof of the symmetry of set identity (Q_1^2 stands for this relation) given the definition of set identity in terms of set membership. This is the problem mentioned by de Champeaux (1979) as not being solvable by his system.

21. $\vdash \neg(Ey)(Ax)(Pxy \leftrightarrow \neg Pxx)$

22. $\vdash (Ey)(Ax)(Pxy \leftrightarrow Pxx) \rightarrow \neg(Ax)(Ey)(Az)(Pxy \leftrightarrow \neg Pzx)$

23. $\vdash (Az)(Ey)(Ax)(Pxy \leftrightarrow (Pxz \& \neg Pxx)) \rightarrow \neg(Ez)(Ax)Pxz$

24. $\vdash \neg(Ey)(Ax)(Pxy \leftrightarrow \neg(Ez)(Pxz \& Pzx))$

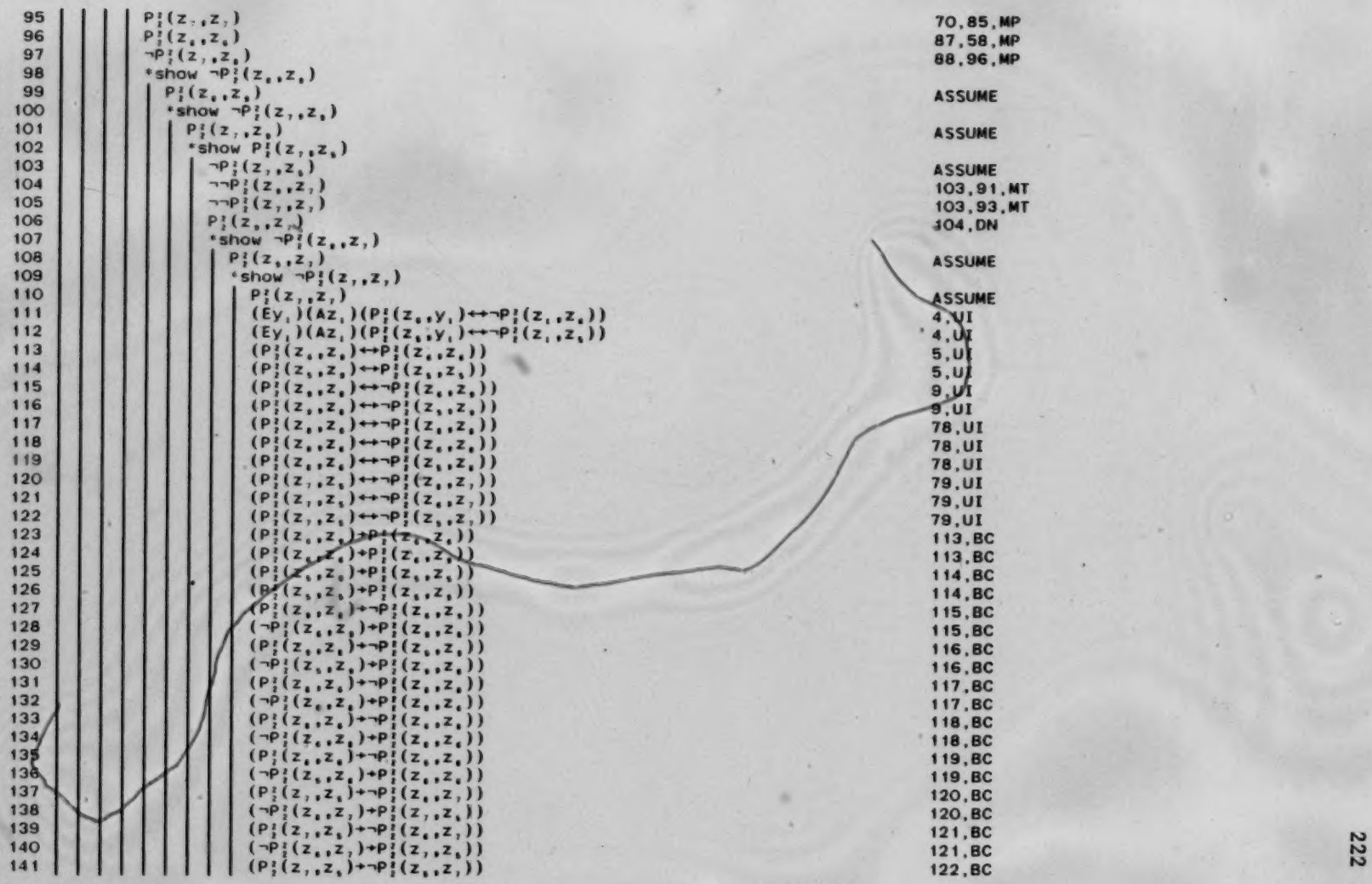
25. $(Au)(Aw)(Quw \leftrightarrow (Az)(Pzu \leftrightarrow Puw)) \vdash (Ax)(Ay)(Qxy \leftrightarrow Qyx)$

```
 1 *show ¬(Ey,)(Ax,)(P²₂(x,,y,)↔¬P²₂(x,,x,))
 2 | (Ey,)(Ax,)(P²₂(x,,y,)↔¬P²₂(x,,x,))                ASSUME
 3 | (Ax,)(P²₂(x,,z,)↔¬P²₂(x,,x,))                     2,EI
 4 | (P²₂(z,,z,)↔¬P²₂(z,,z,))                          3,UI
 5 | (P²₂(z,,z,)→¬P²₂(z,,z,))                          4,BC
 6 | (¬P²₂(z,,z,)→P²₂(z,,z,))                          4,BC
 7 | *show P²₂(z,,z,)
 8 | | ¬P²₂(z,,z,)                                     ASSUME
 9 | | P²₂(z,,z,)                                      8,6,MP
10 | ¬P²₂(z,,z,)                                       5,7,MP
```

CPU time: 73 msec
Statements: 3641

```
1  *show ((Ey1)(Ax1)(P21(x1,y1)↔P21(x1,x1))→¬(Ax1)(Ey1)(Az1)(P21(x1,y1)↔¬P21(z1,x1)))
2  | (Ey1)(Ax1)(P21(x1,y1)↔P21(x1,x1))                                                    ASSUME
3  | *show ¬(Ax1)(Ey1)(Az1)(P21(x1,y1)↔¬P21(z1,x1))
4  | | (Ax1)(Ey1)(Az1)(P21(x1,y1)↔¬P21(z1,x1))                                            ASSUME
5  | | (Ax1)(P21(x1,z1)↔P21(x1,x1))                                                       2,EI
6  | | (Ey1)(Az1)(P21(z1,y1)↔¬P21(z1,z1))                                                 4,UI
7  | | (P21(z1,z1)↔P21(z1,z1))                                                            5,UI
8  | | (P21(z1,z1)→P21(z1,z1))                                                            7,BC
9  | | (Az1)(P21(z1,z1)↔¬P21(z1,z1))                                                      6,EI
10 | | (P21(z1,z1)↔¬P21(z1,z1))                                                           9,UI
11 | | (P21(z1,z1)→¬P21(z1,z1))                                                           10,BC
12 | | (¬P21(z1,z1)→P21(z1,z1))                                                           10,BC
13 | | *show P21(z1,z1)
14 | | | ¬P21(z1,z1)                                                                      ASSUME
15 | | | P21(z1,z1)                                                                       12,14,MP
16 | | | (Ey1)(Az1)(P21(z1,y1)↔¬P21(z1,z1))                                               4,UI
17 | | | (P21(z1,z1)↔P21(z1,z1))                                                          5,UI
18 | | | (P21(z1,z1)↔¬P21(z1,z1))                                                         9,UI
19 | | | (P21(z1,z1)→P21(z1,z1))                                                          17,BC
20 | | | (P21(z1,z1)→P21(z1,z1))                                                          17,BC
21 | | | (P21(z1,z1)→¬P21(z1,z1))                                                         18,BC
22 | | | (¬P21(z1,z1)→P21(z1,z1))                                                         18,BC
23 | | | ¬P21(z1,z1)                                                                      21,15,MP
24 | | | ¬P21(z1,z1)                                                                      23,20,MT
25 | | | (Az1)(P21(z1,z1)↔¬P21(z1,z1))                                                    16,EI
26 | | | (P21(z1,z1)↔¬P21(z1,z1))                                                         25,UI
27 | | | (P21(z1,z1)→¬P21(z1,z1))                                                         26,BC
28 | | | (¬P21(z1,z1)→P21(z1,z1))                                                         26,BC
29 | | | *show P21(z1,z1)
30 | | | | ¬P21(z1,z1)                                                                    ASSUME
31 | | | | *show P21(z1,z1)
32 | | | | | ¬P21(z1,z1)                                                                  ASSUME
33 | | | | | *show P21(z1,z1)
34 | | | | | | ¬P21(z1,z1)                                                                ASSUME
35 | | | | | | ¬¬P21(z1,z1)                                                               34,28,MT
36 | | | | | | *show ¬P21(z1,z1)
37 | | | | | | | P21(z1,z1)                                                               ASSUME
38 | | | | | | | (Ey1)(Az1)(P21(z1,y1)↔¬P21(z1,z1))                                       4,UI
39 | | | | | | | (P21(z1,z1)↔P21(z1,z1))                                                  5,UI
40 | | | | | | | (P21(z1,z1)↔¬P21(z1,z1))                                                 9,UI
41 | | | | | | | (P21(z1,z1)↔¬P21(z1,z1))                                                 25,UI
42 | | | | | | | (P21(z1,z1)↔¬P21(z1,z1))                                                 25,UI
43 | | | | | | | (P21(z1,z1)→P21(z1,z1))                                                  39,BC
44 | | | | | | | (P21(z1,z1)→P21(z1,z1))                                                  39,BC
45 | | | | | | | (P21(z1,z1)→¬P21(z1,z1))                                                 40,BC
46 | | | | | | | (¬P21(z1,z1)→P21(z1,z1))                                                 40,BC
47 | | | | | | | (P21(z1,z1)→¬P21(z1,z1))                                                 41,BC
```

```
48 | | | | | | | | (¬P₂¹(z₂,z₃)→P₂¹(z₃,z₂))                    41,BC
49 | | | | | | | | (P₂¹(z₂,z₂)→¬P₂¹(z₂,z₂))                    42,BC
50 | | | | | | | | (¬P₂¹(z₂,z₃)→P₂¹(z₃,z₂))                    42,BC
51 | | | | | | | | ¬P₂¹(z₂,z₃)                                 45,37,MP
52 | | | | | | | | P₂¹(z₃,z₂)                                  48,32,MP
53 | | | | | | | | ¬P₂¹(z₃,z₂)                                 34,R
54 | | | | | | ¬P₂¹(z₃,z₃)                                   27,33,MP
55 | | | | | P₂¹(z₂,z₃)                                      15,R
56 | | | | ¬P₂¹(z₃,z₃)                                       24,R
57 | | | *show P₂¹(z₃,z₃)
58 | | | | ¬P₂¹(z₃,z₃)                                     ASSUME
59 | | | | ¬¬P₂¹(z₃,z₃)                                    58,12,MT
60 | | | | *show ¬P₂¹(z₃,z₃)
61 | | | | | P₂¹(z₃,z₃)                                    ASSUME
62 | | | | | (Ey₁)(Az₁)(P₂¹(z₃,y₁)↔¬P₂¹(z₁,z₃))            4,UI
63 | | | | | (Ey₁)(Az₁)(P₂¹(z₂,y₁)↔¬P₂¹(z₁,z₂))            4,UI
64 | | | | | (P₂¹(z₃,z₃)↔P₂¹(z₃,z₃))                       5,UI
65 | | | | | (P₂¹(z₂,z₃)↔P₂¹(z₂,z₂))                       5,UI
66 | | | | | (P₂¹(z₃,z₃)↔¬P₂¹(z₃,z₃))                      9,UI
67 | | | | | (P₂¹(z₃,z₃)↔¬P₂¹(z₂,z₃))                      9,UI
68 | | | | | (P₂¹(z₃,z₃)→P₂¹(z₃,z₃))                       64,BC
69 | | | | | (P₂¹(z₃,z₃)→P₂¹(z₃,z₃))                       64,BC
70 | | | | | (P₂¹(z₂,z₃)→P₂¹(z₂,z₂))                       65,BC
71 | | | | | (P₂¹(z₂,z₂)→P₂¹(z₂,z₃))                       65,BC
72 | | | | | (P₂¹(z₃,z₃)→¬P₂¹(z₃,z₃))                      66,BC
73 | | | | | (¬P₂¹(z₃,z₃)→P₂¹(z₃,z₃))                      66,BC
74 | | | | | (P₂¹(z₃,z₃)→¬P₂¹(z₂,z₃))                      67,BC
75 | | | | | (¬P₂¹(z₂,z₃)→P₂¹(z₃,z₃))                      67,BC
76 | | | | | ¬¬P₂¹(z₃,z₃)                                  58,73,MT
77 | | | | | ¬¬P₂¹(z₂,z₃)                                  58,75,MT
78 | | | | | (Az₁)(P₂¹(z₃,z₃)↔¬P₂¹(z₁,z₃))                 62,EI
79 | | | | | (Az₁)(P₂¹(z₂,z₃)↔¬P₂¹(z₁,z₂))                 63,EI
80 | | | | | (P₂¹(z₃,z₃)↔¬P₂¹(z₃,z₂))                      78,UI
81 | | | | | (P₂¹(z₃,z₃)↔¬P₂¹(z₂,z₃))                      78,UI
82 | | | | | (P₂¹(z₂,z₃)↔¬P₂¹(z₃,z₂))                      79,UI
83 | | | | | (P₂¹(z₂,z₃)↔¬P₂¹(z₂,z₂))                      79,UI
84 | | | | | P₂¹(z₃,z₃)                                    76,DN
85 | | | | | P₂¹(z₂,z₃)                                    77,DN
86 | | | | | (P₂¹(z₃,z₃)→¬P₂¹(z₃,z₃))                      80,BC
87 | | | | | (¬P₂¹(z₃,z₃)→P₂¹(z₃,z₃))                      80,BC
88 | | | | | (P₂¹(z₃,z₃)→¬P₂¹(z₂,z₃))                      81,BC
89 | | | | | (¬P₂¹(z₂,z₃)→P₂¹(z₃,z₃))                      81,BC
90 | | | | | (P₂¹(z₂,z₃)→¬P₂¹(z₃,z₂))                      82,BC
91 | | | | | (¬P₂¹(z₃,z₂)→P₂¹(z₂,z₃))                      82,BC
92 | | | | | (P₂¹(z₂,z₃)→¬P₂¹(z₂,z₂))                      83,BC
93 | | | | | (¬P₂¹(z₂,z₂)→P₂¹(z₂,z₃))                      83,BC
94 | | | | | P₂¹(z₃,z₃)                                    68,84,MP
```


95 P₁²(z₇,z₇) 70,85,MP
96 P₁²(z₈,z₈) 87,58,MP
97 ¬P₁²(z₇,z₈) 88,96,MP
98 *show ¬P₁²(z₈,z₈)
99 P₁²(z₈,z₈) ASSUME
100 *show ¬P₁²(z₇,z₈)
101 P₁²(z₇,z₈) ASSUME
102 *show P₁²(z₇,z₈)
103 ¬P₁²(z₇,z₈) ASSUME
104 ¬¬P₁²(z₈,z₇) 103,91,MT
105 ¬¬P₁²(z₇,z₇) 103,93,MT
106 P₁²(z₈,z₇) 104,DN
107 *show ¬P₁²(z₈,z₇)
108 P₁²(z₈,z₇) ASSUME
109 *show ¬P₁²(z₇,z₇)
110 P₁²(z₇,z₇) ASSUME
111 (Ey₁)(Az₁)(P₁²(z₈,y₁)↔¬P₁²(z₁,z₈)) 4,UI
112 (Ey₁)(Az₁)(P₁²(z₈,y₁)↔¬P₁²(z₁,z₈)) 4,UI
113 (P₁²(z₈,z₈)↔P₁²(z₈,z₈)) 5,UI
114 (P₁²(z₈,z₈)↔P₁²(z₈,z₈)) 5,UI
115 (P₁²(z₈,z₈)↔¬P₁²(z₈,z₈)) 9,UI
116 (P₁²(z₇,z₇)↔¬P₁²(z₈,z₈)) 9,UI
117 (P₁²(z₈,z₈)↔¬P₁²(z₈,z₈)) 78,UI
118 (P₁²(z₈,z₇)↔¬P₁²(z₈,z₇)) 78,UI
119 (P₁²(z₈,z₈)↔¬P₁²(z₈,z₈)) 78,UI
120 (P₁²(z₇,z₇)↔¬P₁²(z₈,z₇)) 79,UI
121 (P₁²(z₇,z₈)↔¬P₁²(z₈,z₇)) 79,UI
122 (P₁²(z₇,z₈)↔¬P₁²(z₈,z₇)) 79,UI
123 (P₁²(z₈,z₇)→P₁²(z₇,z₈)) 113,BC
124 (P₁²(z₇,z₈)→P₁²(z₈,z₇)) 113,BC
125 (P₁²(z₈,z₈)→P₁²(z₈,z₈)) 114,BC
126 (P₁²(z₈,z₈)→P₁²(z₈,z₈)) 114,BC
127 (P₁²(z₈,z₈)→¬P₁²(z₈,z₈)) 115,BC
128 (¬P₁²(z₈,z₈)→P₁²(z₈,z₈)) 115,BC
129 (P₁²(z₈,z₈)→¬P₁²(z₈,z₈)) 116,BC
130 (¬P₁²(z₈,z₈)→P₁²(z₈,z₈)) 116,BC
131 (P₁²(z₈,z₈)→¬P₁²(z₈,z₈)) 117,BC
132 (¬P₁²(z₈,z₈)→P₁²(z₈,z₈)) 117,BC
133 (P₁²(z₈,z₈)→¬P₁²(z₈,z₈)) 118,BC
134 (¬P₁²(z₈,z₈)→P₁²(z₈,z₈)) 118,BC
135 (P₁²(z₈,z₈)→¬P₁²(z₈,z₈)) 119,BC
136 (¬P₁²(z₈,z₈)→P₁²(z₈,z₈)) 119,BC
137 (P₁²(z₈,z₈)→¬P₁²(z₈,z₈)) 120,BC
138 (¬P₁²(z₈,z₈)→P₁²(z₈,z₈)) 120,BC
139 (P₁²(z₈,z₈)→¬P₁²(z₈,z₈)) 121,BC
140 (¬P₁²(z₈,z₈)→P₁²(z₈,z₈)) 121,BC
141 (P₁²(z₈,z₈)→¬P₁²(z₈,z₈)) 122,BC

```
142 | | | | | | | | | (¬P¹₁(z₀,z₁)→P¹₁(z₁,z₀))                122,BC
143 | | | | | | | | | ¬P¹₂(z₀,z₀)                          131,96,MP
144 | | | | | | | | | P¹₁(z₀,z₀)                           94,R
145 | | | | | | | | ¬¬P¹₂(z₁,z₁)                         105,R
146 | | | | | | | ¬P¹₂(z₀,z₁)                            90,102,MP
147 | | | | | | | ¬P¹₂(z₁,z₁)                            92,102,MP
148 | | | | | | | P¹₂(z₀,z₁)                             95,R
149 | | | | | ¬¬P¹₂(z₁,z₀)                                77,R
150 | | | ¬P¹₂(z₀,z₀)                                     11,57,MP
```

CPU time: 817 msec
Statements: 60589

```
1  *show ((Az1)(Ey1)(Ax1)(P21(x1,y1)↔(P21(x1,z1)&¬P21(x1,x1)))→¬(Ez1)(Ax1)P21(x1,z1))
2  | (Az1)(Ey1)(Ax1)(P21(x1,y1)↔(P21(x1,z1)&¬P21(x1,x1)))                    ASSUME
3  | *show ¬(Ez1)(Ax1)P21(x1,z1)
4  | | (Ez1)(Ax1)P21(x1,z1)                                                  ASSUME
5  | | (Ax1)P21(x1,z2)                                                      4,EI
6  | | (Ey1)(Ax1)(P21(x1,y1)↔(P21(x1,z2)&¬P21(x1,x1)))                      2,UI
7  | | P21(z2,z2)                                                           5,UI
8  | | (Ax1)(P21(x1,z2)↔(P21(x1,z2)&¬P21(x1,x1)))                           6,EI
9  | | (P21(z2,z2)↔(P21(z2,z2)&¬P21(z2,z2)))                                8,UI
10 | | (P21(z2,z2)→(P21(z2,z2)&¬P21(z2,z2)))                                9,BC
11 | | ((P21(z2,z2)&¬P21(z2,z2))→P21(z2,z2))                                9,BC
12 | | *show P21(z2,z2)
13 | | | ¬P21(z2,z2)                                                        ASSUME
14 | | | ¬(P21(z2,z2)&¬P21(z2,z2))                                          13,11,MT
15 | | | *show (P21(z2,z2)&¬P21(z2,z2))
16 | | | | P21(z2,z2)                                                       7,R
17 | | | | *show ¬P21(z2,z2)
18 | | | | | P21(z2,z2)                                                     ASSUME
19 | | | | | (Ey1)(Ax1)(P21(x1,y1)↔(P21(x1,z2)&¬P21(x1,x1)))                2,UI
20 | | | | | P21(z2,z2)                                                     5,UI
21 | | | | | (P21(z2,z2)↔(P21(z2,z2)&¬P21(z2,z2)))                          8,UI
22 | | | | | (P21(z2,z2)→(P21(z2,z2)&¬P21(z2,z2)))                          21,BC
23 | | | | | ((P21(z2,z2)&¬P21(z2,z2))→P21(z2,z2))                          21,BC
24 | | | | | (Ax1)(P21(x1,z2)↔(P21(x1,z2)&¬P21(x1,x1)))                     19,EI
25 | | | | | (P21(z2,z2)↔(P21(z2,z2)&¬P21(z2,z2)))                          24,UI
26 | | | | | (P21(z2,z2)→(P21(z2,z2)&¬P21(z2,z2)))                          25,BC
27 | | | | | ((P21(z2,z2)&¬P21(z2,z2))→P21(z2,z2))                          25,BC
28 | | | | | *show P21(z2,z2)
29 | | | | | | ¬P21(z2,z2)                                                  ASSUME
30 | | | | | | ¬(P21(z2,z2)&¬P21(z2,z2))                                    29,23,MT
31 | | | | | | (P21(z2,z2)&¬P21(z2,z2))                                     20,29,ADJ
32 | | | | | (P21(z2,z2)&¬P21(z2,z2))                                       22,28,MP
33 | | | | | ¬P21(z2,z2)                                                    32,S
34 | | | | (P21(z2,z2)&¬P21(z2,z2))                                         16,17,ADJ
35 | | | (P21(z2,z2)&¬P21(z2,z2))                                           10,12,MP
36 | | ¬P21(z2,z2)                                                          35,S

CPU time: 204 msec
Statements: 16772
```

```
1  *show ¬(Ey1)(Ax1)(P21(x1,y1)↔¬(Ez1)(P21(x1,z1)&P21(z1,x1)))
2  | (Ey1)(Ax1)(P21(x1,y1)↔¬(Ez1)(P21(x1,z1)&P21(z1,x1)))          ASSUME
3  | (Ax1)(P21(x1,z0)↔¬(Ez1)(P21(x1,z1)&P21(z1,x1)))               2,EI
4  | (P21(z0,z0)↔¬(Ez1)(P21(z0,z1)&P21(z1,z0)))                   3,UI
5  | (P21(z0,z0)→¬(Ez1)(P21(z0,z1)&P21(z1,z0)))                   4,BC
6  | (¬(Ez1)(P21(z0,z1)&P21(z1,z0))→P21(z0,z0))                   4,BC
7  | *show P21(z0,z0)
8  | | ¬P21(z0,z0)                                                ASSUME
9  | | ¬¬(Ez1)(P21(z0,z1)&P21(z1,z0))                            8,6,MT
10 | | (Ez1)(P21(z0,z1)&P21(z1,z0))                              9,DN
11 | | (P21(z0,z0)&P21(z0,z0))                                   10,EI
12 | | P21(z0,z0)                                                11,S
13 | | P21(z0,z0)                                                11,S
14 | | *show ¬(Ez1)(P21(z0,z1)&P21(z1,z0))
15 | | | (Ez1)(P21(z0,z1)&P21(z1,z0))                            ASSUME
16 | | | (P21(z0,z0)↔¬(Ez1)(P21(z0,z1)&P21(z1,z0)))              3,UI
17 | | | (P21(z0,z0)→¬(Ez1)(P21(z0,z1)&P21(z1,z0)))              16,BC
18 | | | (¬(Ez1)(P21(z0,z1)&P21(z1,z0))→P21(z0,z0))              16,BC
19 | | | ¬(Ez1)(P21(z0,z1)&P21(z1,z0))                           17,13,MP
20 | | | (Az1)¬(P21(z0,z1)&P21(z1,z0))                           19,QN
21 | | | ¬(P21(z0,z0)&P21(z0,z0))                                20,UI
22 | | | (P21(z0,z0)&P21(z0,z0))                                 13,12,ADJ
23 | ¬(Ez1)(P21(z0,z1)&P21(z1,z0))                               5,7,MP
24 | (Az1)¬(P21(z0,z1)&P21(z1,z0))                               23,QN
25 | ¬(P21(z0,z0)&P21(z0,z0))                                    24,UI
26 | (P21(z0,z0)&P21(z0,z0))                                     7,7,ADJ

CPU time: 191 msec
Statements: 15153
```

```
1  *show (Ax1)(Ay1)(Q21(x1,y1)↔Q21(y1,x1))
2  | *show (Ay1)(Q21(x1,y1)↔Q21(y1,x1))
3  | | *show (Q21(x1,y1)↔Q21(y1,x1))
4  | | | *show (Q21(y1,x1)→Q21(x1,y1))
5  | | | | Q21(y1,x1)                                                    ASSUME
6  | | | | *show Q21(x1,y1)
7  | | | | | ¬Q21(x1,y1)                                                 ASSUME
8  | | | | | (Az1)(Aw1)(Q21(z1,w1)↔(Az2)(P21(z2,z1)↔P21(z2,w1)))         PREM
9  | | | | | (Aw1)(Q21(y1,w1)↔(Az2)(P21(z2,y1)↔P21(z2,w1)))              8,UI
10 | | | | | (Aw1)(Q21(x1,w1)↔(Az2)(P21(z2,x1)↔P21(z2,w1)))              8,UI
11 | | | | | (Q21(y1,y1)↔(Az2)(P21(z2,y1)↔P21(z2,y1)))                   9,UI
12 | | | | | (Q21(y1,x1)↔(Az2)(P21(z2,y1)↔P21(z2,x1)))                   9,UI
13 | | | | | (Q21(x1,y1)↔(Az2)(P21(z2,x1)↔P21(z2,y1)))                   10,UI
14 | | | | | (Q21(x1,x1)↔(Az2)(P21(z2,x1)↔P21(z2,x1)))                   10,UI
15 | | | | | (Q21(y1,y1)→(Az2)(P21(z2,y1)↔P21(z2,y1)))                   11,BC
16 | | | | | ((Az2)(P21(z2,y1)↔P21(z2,y1))→Q21(y1,y1))                   11,BC
17 | | | | | (Q21(y1,x1)→(Az2)(P21(z2,y1)↔P21(z2,x1)))                   12,BC
18 | | | | | ((Az2)(P21(z2,y1)↔P21(z2,x1))→Q21(y1,x1))                   12,BC
19 | | | | | (Q21(x1,y1)→(Az2)(P21(z2,x1)↔P21(z2,y1)))                   13,BC
20 | | | | | ((Az2)(P21(z2,x1)↔P21(z2,y1))→Q21(x1,y1))                   13,BC
21 | | | | | (Q21(x1,x1)→(Az2)(P21(z2,x1)↔P21(z2,x1)))                   14,BC
22 | | | | | ((Az2)(P21(z2,x1)↔P21(z2,x1))→Q21(x1,x1))                   14,BC
23 | | | | | (Az2)(P21(z2,y1)↔P21(z2,x1))                                17,5,MP
24 | | | | | ¬(Az2)(P21(z2,x1)↔P21(z2,y1))                               7,20,MT
25 | | | | | (Ez2)¬(P21(z2,x1)↔P21(z2,y1))                               24,QN
26 | | | | | (P21(y1,y1)↔P21(y1,x1))                                     23,UI
27 | | | | | (P21(x1,y1)↔P21(x1,x1))                                     23,UI
28 | | | | | (P21(y1,y1)→P21(y1,x1))                                     26,BC
29 | | | | | (P21(y1,x1)→P21(y1,y1))                                     26,BC
30 | | | | | (P21(x1,y1)→P21(x1,x1))                                     27,BC
31 | | | | | (P21(x1,x1)→P21(x1,y1))                                     27,BC
32 | | | | | ¬(P21(z1,x1)↔P21(z1,y1))                                    25,EI
33 | | | | | *show (P21(z1,x1)↔P21(z1,y1))
34 | | | | | | *show (P21(z1,y1)→P21(z1,x1))
35 | | | | | | | P21(z1,y1)                                            ASSUME
36 | | | | | | | *show P21(z1,x1)
37 | | | | | | | | ¬P21(z1,x1)                                         ASSUME
38 | | | | | | | | *show Q21(y1,y1)
39 | | | | | | | | | ¬Q21(y1,y1)                                       ASSUME
40 | | | | | | | | | ¬(Az2)(P21(z2,y1)↔P21(z2,y1))                     39,16,MT
41 | | | | | | | | | (Ez2)¬(P21(z2,y1)↔P21(z2,y1))                     40,QN
42 | | | | | | | | | ¬(P21(z1,y1)↔P21(z1,y1))                          41,EI
43 | | | | | | | | | *show (P21(z1,y1)↔P21(z1,y1))
44 | | | | | | | | | | *show (P21(z1,y1)→P21(z1,y1))
45 | | | | | | | | | | | P21(z1,y1)                                  ASSUME
46 | | | | | | | | | | (P21(z1,y1)↔P21(z1,y1))                       44,44,CB
47 | | | | | | | | (Az2)(P21(z2,y1)↔P21(z2,y1))                        15,38,MP
```

```
48 | | | | | | | | (P¹₁(y₁,y₁)↔P¹₁(y₁,y₁))                                                                  47,UI
49 | | | | | | | | (P¹₂(x₁,y₁)↔P¹₂(x₁,y₁))                                                                  47,UI
50 | | | | | | | | (P¹₁(y₁,y₁)→P¹₁(y₁,y₁))                                                                  48,BC
51 | | | | | | | | (P¹₂(x₁,y₁)→P¹₂(x₁,y₁))                                                                  49,BC
52 | | | | | | | | *show (Az₂)(P¹₂(z₂,x₁)↔P¹₂(z₂,y₁))
53 | | | | | | | |  *show (P¹₂(z₂,x₁)↔P¹₂(z₂,y₁))
54 | | | | | | | |  | *show (P¹₂(z₂,y₁)→P¹₂(z₂,x₁))
55 | | | | | | | |  | | P¹₂(z₂,y₁)                                                                          ASSUME
56 | | | | | | | |  | | *show P¹₂(z₂,x₁)
57 | | | | | | | |  | | | ¬P¹₂(z₂,x₁)                                                                       ASSUME
58 | | | | | | | |  | | | (Az₁)(Aw₁)(Q¹₂(z₁,w₁)↔(Az₂)(P¹₂(z₂,z₁)↔P¹₂(z₂,w₁)))                                 PREM
59 | | | | | | | |  | | | (Aw₁)(Q¹₂(z₁,w₁)↔(Az₂)(P¹₂(z₂,z₁)↔P¹₂(z₂,w₁)))                                       58,UI
60 | | | | | | | |  | | | (Q¹₁(y₁,z₁)↔(Az₂)(P¹₂(z₂,y₁)↔P¹₂(z₂,z₁)))                                             9,UI
61 | | | | | | | |  | | | (Q¹₂(x₁,z₁)↔(Az₂)(P¹₂(z₂,x₁)↔P¹₂(z₂,z₁)))                                             10,UI
62 | | | | | | | |  | | | (P¹₂(z₂,y₁)↔P¹₂(z₂,x₁))                                                               23,UI
63 | | | | | | | |  | | | P¹₂(z₂,x₁)                                                                            62,BC,55,MP
64 | | | | | | | |  | *show (P¹₂(z₂,x₁)→P¹₂(z₂,y₁))
65 | | | | | | | |  | | P¹₂(z₂,x₁)                                                                          ASSUME
66 | | | | | | | |  | | *show P¹₂(z₂,y₁)
67 | | | | | | | |  | | | ¬P¹₂(z₂,y₁)                                                                       ASSUME
68 | | | | | | | |  | | | (Az₁)(Aw₁)(Q¹₂(z₁,w₁)↔(Az₂)(P¹₂(z₂,z₁)↔P¹₂(z₂,w₁)))                                 PREM
69 | | | | | | | |  | | | (Aw₁)(Q¹₂(z₁,w₁)↔(Az₂)(P¹₂(z₂,z₁)↔P¹₂(z₂,w₁)))                                       68,UI
70 | | | | | | | |  | | | (Q¹₁(y₁,z₁)↔(Az₂)(P¹₂(z₂,y₁)↔P¹₂(z₂,z₁)))                                             9,UI
71 | | | | | | | |  | | | (Q¹₂(x₁,z₁)↔(Az₂)(P¹₂(z₂,x₁)↔P¹₂(z₂,z₁)))                                             10,UI
72 | | | | | | | |  | | | (P¹₂(z₂,y₁)↔P¹₂(z₂,x₁))                                                               23,UI
73 | | | | | | | |  | | | P¹₂(z₂,y₁)                                                                            72,BC,65,MP
74 | | | | | | | |  | | (P¹₂(z₂,x₁)↔P¹₂(z₂,y₁))                                                                64,54,CB
75 | | | | | | | | ¬(Az₂)(P¹₂(z₂,x₁)↔P¹₂(z₂,y₁))                                                           24,R
76 | | | | | | | | *show (P¹₂(z₂,x₁)→P¹₂(z₂,y₁))
77 | | | | | | | |  P¹₂(z₂,x₁)                                                                            ASSUME
78 | | | | | | | |  *show P¹₂(z₂,y₁)
79 | | | | | | | |  | ¬P¹₂(z₂,y₁)                                                                         ASSUME
80 | | | | | | | |  | *show Q¹₁(y₁,y₁)
81 | | | | | | | |  | | ¬Q¹₁(y₁,y₁)                                                                       ASSUME
82 | | | | | | | |  | | ¬(Az₂)(P¹₂(z₂,y₁)↔P¹₂(z₂,y₁))                                                       81,16,MT
83 | | | | | | | |  | | (Ez₂)¬(P¹₂(z₂,y₁)↔P¹₂(z₂,y₁))                                                       82,QN
84 | | | | | | | |  | | ¬(P¹₂(z₂,y₁)↔P¹₂(z₂,y₁))                                                            83,EI
85 | | | | | | | |  | | *show (P¹₂(z₂,y₁)↔P¹₂(z₂,y₁))
86 | | | | | | | |  | | | *show (P¹₂(z₂,y₁)→P¹₂(z₂,y₁))
87 | | | | | | | |  | | | | P¹₂(z₂,y₁)                                                                      ASSUME
88 | | | | | | | |  | | | (P¹₂(z₂,y₁)↔P¹₂(z₂,y₁))                                                           86,86,CB
89 | | | | | | | |  | | (Az₂)(P¹₂(z₂,y₁)↔P¹₂(z₂,y₁))                                                         15,80,MP
90 | | | | | | | |  | | (P¹₁(y₁,y₁)↔P¹₁(y₁,y₁))                                                             89,UI
91 | | | | | | | |  | | (P¹₂(x₁,y₁)↔P¹₂(x₁,y₁))                                                             89,UI
92 | | | | | | | |  | | (P¹₁(y₁,y₁)→P¹₁(y₁,y₁))                                                             90,BC
93 | | | | | | | |  | | (P¹₂(x₁,y₁)→P¹₂(x₁,y₁))                                                             91,BC
94 | | | | | | | |  | *show (Az₂)(P¹₂(z₂,x₁)↔P¹₂(z₂,y₁))
```

```
95  | | | | | | | | | *show (P₁²(z₂,x₁)↔P₁²(z₂,y₁))                                                
```

```
142 | | | | (P₁²(x₁,x₁)↔P₁²(x₁,y₁))                                                                                    138,UI
143 | | | | (P₁²(y₁,x₁)→P₁²(y₁,y₁))                                                                                    141,BC
144 | | | | (P₁²(y₁,y₁)→P₁²(y₁,x₁))                                                                                    141,BC
145 | | | | (P₁²(x₁,x₁)→P₁²(x₁,y₁))                                                                                    142,BC
146 | | | | (P₁²(x₁,y₁)→P₁²(x₁,x₁))                                                                                    142,BC
147 | | | | ¬(P₁²(z₀,y₁)↔P₁²(z₀,x₁))                                                                                   140,EI
148 | | | | *show (P₁²(z₀,y₁)↔P₁²(z₀,x₁))
149 | | | |  *show (P₁²(z₀,x₁)→P₁²(z₀,y₁))
150 | | | |   P₁²(z₀,x₁)                                                                                             ASSUME
151 | | | |   *show P₁²(z₀,y₁)
152 | | | |    ¬P₁²(z₀,y₁)                                                                                           ASSUME
153 | | | |    *show Q₁²(y₁,y₁)
154 | | | |     ¬Q₁²(y₁,y₁)                                                                                          ASSUME
155 | | | |     ¬(Ax₁)(P₁²(z₀,y₁)↔P₁²(z₀,y₁))                                                                        154,131,MT
156 | | | |     (Ex₁)¬(P₁²(z₀,y₁)↔P₁²(z₀,y₁))                                                                        155,QN
157 | | | |     ¬(P₁²(z₀,y₁)↔P₁²(z₀,y₁))                                                                             156,EI
158 | | | |     *show (P₁²(z₀,y₁)↔P₁²(z₀,y₁))
159 | | | |      *show (P₁²(z₀,y₁)→P₁²(z₀,y₁))
160 | | | |       P₁²(z₀,y₁)                                                                                        ASSUME
161 | | | |      (P₁²(z₀,y₁)↔P₁²(z₀,y₁))                                                                             159,159,CB
162 | | | |    (Ax₁)(P₁²(z₀,y₁)↔P₁²(z₀,y₁))                                                                           130,153,MP
163 | | | |    (P₁²(y₁,y₁)↔P₁²(y₁,y₁))                                                                               162,UI
164 | | | |    (P₁²(x₁,y₁)↔P₁²(x₁,y₁))                                                                               162,UI
165 | | | |    (P₁²(y₁,y₁)→P₁²(y₁,y₁))                                                                               163,BC
166 | | | |    (P₁²(x₁,y₁)→P₁²(x₁,y₁))                                                                               164,BC
167 | | | |    *show (Ax₁)(P₁²(z₀,y₁)↔P₁²(z₀,x₁))
168 | | | |     *show (P₁²(z₀,y₁)↔P₁²(z₀,x₁))
169 | | | |      *show (P₁²(z₀,x₁)→P₁²(z₀,y₁))
170 | | | |       P₁²(z₀,x₁)                                                                                        ASSUME
171 | | | |       *show P₁²(z₀,y₁)
172 | | | |        ¬P₁²(z₀,y₁)                                                                                      ASSUME
173 | | | |        (Ax₁)(Aw₁)(Q₁²(z₁,w₁)↔(Ax₂)(P₁²(z₂,z₁)↔P₁²(z₂,w₁)))                                                 PREM
174 | | | |        (Aw₁)(Q₁²(z₀,w₁)↔(Ax₂)(P₁²(z₂,z₂)↔P₁²(z₂,w₁)))                                                   173,UI
175 | | | |        (Q₁²(y₁,z₂)↔(Ax₂)(P₁²(z₂,y₁)↔P₁²(z₂,z₂)))                                                       124,UI
176 | | | |        (Q₁²(x₁,z₂)↔(Ax₂)(P₁²(z₂,x₁)↔P₁²(z₂,z₂)))                                                       125,UI
177 | | | |        (P₁²(z₀,x₁)↔P₁²(z₀,y₁))                                                                          138,UI
178 | | | |        P₁²(z₀,y₁)                                                                                       177,BC,170,MP
179 | | | |       *show (P₁²(z₀,y₁)→P₁²(z₀,x₁))
180 | | | |        P₁²(z₀,y₁)                                                                                       ASSUME
181 | | | |        *show P₁²(z₀,x₁)
182 | | | |         ¬P₁²(z₀,x₁)                                                                                     ASSUME
183 | | | |         (Ax₁)(Aw₁)(Q₁²(z₁,w₁)↔(Ax₂)(P₁²(z₂,z₁)↔P₁²(z₂,w₁)))                                            PREM
184 | | | |         (Aw₁)(Q₁²(z₀,w₁)↔(Ax₂)(P₁²(z₂,z₂)↔P₁²(z₂,w₁)))                                                 183,UI
185 | | | |         (Q₁²(y₁,z₂)↔(Ax₂)(P₁²(z₂,y₁)↔P₁²(z₂,z₂)))                                                     124,UI
186 | | | |         (Q₁²(x₁,z₂)↔(Ax₂)(P₁²(z₂,x₁)↔P₁²(z₂,z₂)))                                                     125,UI
187 | | | |         (P₁²(z₀,x₁)↔P₁²(z₀,y₁))                                                                        138,UI
188 | | | |         P₁²(z₀,x₁)                                                                                      187,BC,180,MP
```

```
189 | | | | | | | | | | (P¦(z₂,y₁)↔P¦(z₂,x₁))                                                                  179,169,CB
190 | | | | | | | | | ¬(Az₂)(P¦(z₂,y₁)↔P¦(z₂,x₁))                                                            139,R
191 | | | | | | | *show (P¦(z₀,y₁)→P¦(z₀,x₁))
192 | | | | | | |  P¦(z₀,y₁)                                                                            ASSUME
193 | | | | | | |  *show P¦(z₀,x₁)
194 | | | | | | |  | ¬P¦(z₀,x₁)                                                                          ASSUME
195 | | | | | | |  | *show Q¦(y₁,y₁)
196 | | | | | | |  | | ¬Q¦(y₁,y₁)                                                                        ASSUME
197 | | | | | | |  | | ¬(Az₂)(P¦(z₂,y₁)↔P¦(z₂,y₁))                                                        196,131,MT
198 | | | | | | |  | | (Ez₂)¬(P¦(z₂,y₁)↔P¦(z₂,y₁))                                                        197,QN
199 | | | | | | |  | | ¬(P¦(z₀,y₁)↔P¦(z₀,y₁))                                                             198,EI
200 | | | | | | |  | | *show (P¦(z₀,y₁)↔P¦(z₀,y₁))
201 | | | | | | |  | | | *show (P¦(z₀,y₁)→P¦(z₀,y₁))
202 | | | | | | |  | | | | P¦(z₀,y₁)                                                                       ASSUME
203 | | | | | | |  | | | (P¦(z₀,y₁)↔P¦(z₀,y₁))                                                           201,201,CB
204 | | | | | | |  | (Az₂)(P¦(z₂,y₁)↔P¦(z₂,y₁))                                                           130,195,MP
205 | | | | | | |  | (P¦(y₀,y₁)↔P¦(y₀,y₁))                                                               204,UI
206 | | | | | | |  | (P¦(x₀,y₁)↔P¦(x₀,y₁))                                                               204,UI
207 | | | | | | |  | (P¦(y₀,y₁)→P¦(y₀,y₁))                                                               205,BC
208 | | | | | | |  | (P¦(x₀,y₁)→P¦(x₀,y₁))                                                               206,BC
209 | | | | | | |  | *show (Az₂)(P¦(z₂,y₁)↔P¦(z₂,x₁))
210 | | | | | | |  | | *show (P¦(z₀,y₁)↔P¦(z₀,x₁))
211 | | | | | | |  | | | *show (P¦(z₀,x₁)→P¦(z₀,y₁))
212 | | | | | | |  | | | | P¦(z₀,x₁)                                                                     ASSUME
213 | | | | | | |  | | | | *show P¦(z₀,y₁)
214 | | | | | | |  | | | | | ¬P¦(z₀,y₁)                                                                  ASSUME
215 | | | | | | |  | | | | | (Az₁)(Aw₁)(Q¦(z₁,w₁)↔(Az₂)(P¦(z₂,z₁)↔P¦(z₂,w₁)))                                PREM
216 | | | | | | |  | | | | | (Aw₁)(Q¦(z₀,w₁)↔(Az₂)(P¦(z₂,z₁)↔P¦(z₂,w₁)))                                      215,UI
217 | | | | | | |  | | | | | (Q¦(y₁,z₁)↔(Az₂)(P¦(z₂,y₁)↔P¦(z₂,z₁)))                                           124,UI
218 | | | | | | |  | | | | | (Q¦(x₁,z₁)↔(Az₂)(P¦(z₂,x₁)↔P¦(z₂,z₁)))                                           125,UI
219 | | | | | | |  | | | | | (P¦(z₀,x₁)↔P¦(z₀,y₁))                                                            138,UI
220 | | | | | | |  | | | | | P¦(z₀,y₁)                                                                       219,BC,212,MP
221 | | | | | | |  | | | *show (P¦(z₀,y₁)→P¦(z₀,x₁))
222 | | | | | | |  | | | | P¦(z₀,y₁)                                                                     ASSUME
223 | | | | | | |  | | | | *show P¦(z₀,x₁)
224 | | | | | | |  | | | | | ¬P¦(z₀,x₁)                                                                  ASSUME
225 | | | | | | |  | | | | | (Az₁)(Aw₁)(Q¦(z₁,w₁)↔(Az₂)(P¦(z₂,z₁)↔P¦(z₂,w₁)))                                PREM
226 | | | | | | |  | | | | | (Aw₁)(Q¦(z₀,w₁)↔(Az₂)(P¦(z₂,z₁)↔P¦(z₂,w₁)))                                      225,UI
227 | | | | | | |  | | | | | (Q¦(y₁,z₁)↔(Az₂)(P¦(z₂,y₁)↔P¦(z₂,z₁)))                                           124,UI
228 | | | | | | |  | | | | | (Q¦(x₁,z₁)↔(Az₂)(P¦(z₂,x₁)↔P¦(z₂,z₁)))                                           125,UI
229 | | | | | | |  | | | | | (P¦(z₀,x₁)↔P¦(z₀,y₁))                                                            138,UI
230 | | | | | | |  | | | | | P¦(z₀,x₁)                                                                       229,BC,222,MP
231 | | | | | | |  | | | (P¦(z₀,y₁)↔P¦(z₀,x₁))                                                             221,211,CB
232 | | | | | | |  | ¬(Az₂)(P¦(z₂,y₁)↔P¦(z₂,x₁))                                                           139,R
233 | | | | | | (P¦(z₀,y₁)↔P¦(z₀,x₁))                                                                191,149,CB
234 | | | | | (Q¦(x₁,y₁)↔Q¦(y₁,x₁))                                                                  119,4,CB
```

CPU time: 558 msec
Statements: 40301

In Chapter VII, Section B, three different versions of an equivalence which could not be proved by Bledsoe's systems were mentioned. They are:

26. $\vdash (((p\&(q\rightarrow r))\rightarrow s)\leftrightarrow((\neg p+(q+s))\&(\neg p+(\neg r+s))))$

27. $\vdash (Ax)((Pa\&(Px\rightarrow Pb))\rightarrow Pc) \leftrightarrow$
$(Ex)((\neg Pa+((Px+Pc))\&(\neg Pa+(\neg Pb+Pc)))$

28. $\vdash ((Ax)((Pa\&(Px\rightarrow(Ey)(Py\&Rxy)))\rightarrow(Ez)(Ew)(Pz\&Rxw\&Rwz))$
$\leftrightarrow (Ax)((\neg Pa+(Px+(Ez)(Ew)(Pz\&Rxw\&Rwz))) \&$
$(\neg Pa+(\neg(Ey)(Py\&Rxy)+(Ez)(Ew)(Pz\&Rxw\&Rwz)))))$

(The line numbering on the proof of (28) is a bit peculiar because lines were so long as to require breaking across boundries. The proof line number ended up on the same line of the page as the annotation. Thus for example line 12 of the proof actually starts one line earlier on the page than the line number would seem to indicate.)

```
1  *show (((p&(q+r))→s)↔((¬p+(q+s))&(¬p+(¬r+s))))
2  | *show (((¬p+(q+s))&(¬p+(¬r+s)))→((p&(q+r))→s))
3  | | ((¬p+(q+s))&(¬p+(¬r+s)))                          ASSUME
4  | | *show ((p&(q+r))→s)
5  | | | (p&(q+r))                                       ASSUME
6  | | | *show s
7  | | | | ¬s                                           ASSUME
8  | | | | (¬p+(q+s))                                   3,S
9  | | | | (¬p+(¬r+s))                                  3,S
10 | | | | p                                            5,S
11 | | | | (q+r)                                        5,S
12 | | | | (q+s)                                        10,8,MTP
13 | | | | (¬r+s)                                       10,9,MTP
14 | | | | q                                            7,12,MTP
15 | | | | ¬r                                           7,13,MTP
16 | | | | r                                            11,14,MP
17 *show (((p&(q+r))→s)→((¬p+(q+s))&(¬p+(¬r+s))))
18 | ((p&(q+r))→s)                                      ASSUME
19 | *show ((¬p+(q+s))&(¬p+(¬r+s)))
20 | | *show (¬p+(q+s))
21 | | | ¬(¬p+(q+s))                                    ASSUME
22 | | | *show ¬p
23 | | | | p                                            ASSUME
24 | | | | *show (q+s)
25 | | | | | ¬(q+s)                                     ASSUME
26 | | | | | *show q
27 | | | | | | ¬q                                       ASSUME
28 | | | | | | *show s
29 | | | | | | | ¬s                                     ASSUME
30 | | | | | | | ¬(p&(q+r))                             29,18,MT
31 | | | | | | | *show (p&(q+r))
32 | | | | | | | | p                                    23,R
33 | | | | | | | | *show (q+r)
34 | | | | | | | | | q                                  ASSUME
35 | | | | | | | | | ¬q                                 27,R
36 | | | | | | | | (p&(q+r))                          32,33,ADJ
37 | | | | | | ¬(q+s)                                   25,R
38 | | | | | | (q+s)                                    28,ADD
39 | | | | | (q+s)                                      26,ADD
40 | | | | ¬(¬p+(q+s))                                  21,R
41 | | | | (¬p+(q+s))                                   24,ADD
42 | | | (¬p+(q+s))                                     22,ADD
43 | | *show (¬p+(¬r+s))
44 | | | ¬(¬p+(¬r+s))                                   ASSUME
45 | | | *show ¬p
46 | | | | p                                            ASSUME
47 | | | | (q+s)                                        46,20,MTP
```

```
48 | | | | | *show (¬r+s)
49 | | | | | | ¬(¬r+s)                                          ASSUME
50 | | | | | | *show ¬r
51 | | | | | | | r                                          ASSUME
52 | | | | | | | *show s
53 | | | | | | | | ¬s                                       ASSUME
54 | | | | | | | | ¬(p&(q+r))                               53,18,MT
55 | | | | | | | | q                                        53,47,MTP
56 | | | | | | | | *show (p&(q+r))
57 | | | | | | | | | p                                      46,R
58 | | | | | | | | | *show (q+r)
59 | | | | | | | | | | r                                    51,R
60 | | | | | | | | | (p&(q+r))                              57,58,ADJ
61 | | | | | | | | ¬(¬r+s)                                  49,R
62 | | | | | | | | (¬r+s)                                   52,ADD
63 | | | | | | | (¬r+s)                                     50,ADD
64 | | | | | | ¬(¬p+(¬r+s))                                 44,R
65 | | | | | | (¬p+(¬r+s))                                  48,ADD
66 | | | | | (¬p+(¬r+s))                                    45,ADD
67 | | | | ((¬p+(q+s))&(¬p+(¬r+s)))                         20,43,ADJ
68 | | | (((p&(q+r))+s)↔((¬p+(q+s))&(¬p+(¬r+s))))           17,2,CB
```

CPU time: 212 msec
Statements: 17527

```
1  *show ((Ax1)((P¦(a1)&(P¦(x1)→P¦(b1)))→P¦(c1))↔(Ax1)((¬P¦(a1)+(P¦(x1)+P¦(c1)))&(¬P¦(a1)+(¬P¦(b1)+P¦(c1)))))
2  | *show ((Ax1)((¬P¦(a1)+(P¦(x1)+P¦(c1)))&(¬P¦(a1)+(¬P¦(b1)+P¦(c1))))→(Ax1)((P¦(a1)&(P¦(x1)→P¦(b1)))→P¦(c1)))
3  | | (Ax1)((¬P¦(a1)+(P¦(x1)+P¦(c1)))&(¬P¦(a1)+(¬P¦(b1)+P¦(c1))))                                                ASSUME
4  | | *show (Ax1)((P¦(a1)&(P¦(x1)→P¦(b1)))→P¦(c1))
5  | | | *show ((P¦(a1)&(P¦(x1)→P¦(b1)))→P¦(c1))
6  | | | | (P¦(a1)&(P¦(x1)→P¦(b1)))                                                                               ASSUME
7  | | | | *show P¦(c1)
8  | | | | | ¬P¦(c1)                                                                                              ASSUME
9  | | | | | P¦(a1)                                                                                               6,S
10 | | | | | (P¦(x1)→P¦(b1))                                                                                      6,S
11 | | | | | ((¬P¦(a1)+(P¦(x1)+P¦(c1)))&(¬P¦(a1)+(¬P¦(b1)+P¦(c1))))                                                3,UI
12 | | | | | ((¬P¦(a1)+(P¦(a1)+P¦(c1)))&(¬P¦(a1)+(¬P¦(b1)+P¦(c1))))                                                3,UI
13 | | | | | ((¬P¦(a1)+(P¦(c1)+P¦(c1)))&(¬P¦(a1)+(¬P¦(b1)+P¦(c1))))                                                3,UI
14 | | | | | ((¬P¦(a1)+(P¦(b1)+P¦(c1)))&(¬P¦(a1)+(¬P¦(b1)+P¦(c1))))                                                3,UI
15 | | | | | (¬P¦(a1)+(P¦(x1)+P¦(c1)))                                                                            11,S
16 | | | | | (¬P¦(a1)+(¬P¦(b1)+P¦(c1)))                                                                           11,S
17 | | | | | (¬P¦(a1)+(P¦(a1)+P¦(c1)))                                                                            12,S
18 | | | | | (¬P¦(a1)+(P¦(c1)+P¦(c1)))                                                                            13,S
19 | | | | | (¬P¦(a1)+(P¦(b1)+P¦(c1)))                                                                            14,S
20 | | | | | (P¦(x1)+P¦(c1))                                                                                      9,15,MTP
21 | | | | | (¬P¦(b1)+P¦(c1))                                                                                     9,16,MTP
22 | | | | | (P¦(a1)+P¦(c1))                                                                                      9,17,MTP
23 | | | | | (P¦(c1)+P¦(c1))                                                                                      9,18,MTP
24 | | | | | P¦(c1)                                                                                               23,8,MTP
25 | *show ((Ax1)((P¦(a1)&(P¦(x1)→P¦(b1)))→P¦(c1))→(Ax1)((¬P¦(a1)+(P¦(x1)+P¦(c1)))&(¬P¦(a1)+(¬P¦(b1)+P¦(c1)))))
26 | | (Ax1)((P¦(a1)&(P¦(x1)→P¦(b1)))→P¦(c1))                                                                     ASSUME
27 | | *show (Ax1)((¬P¦(a1)+(P¦(x1)+P¦(c1)))&(¬P¦(a1)+(¬P¦(b1)+P¦(c1))))
28 | | | *show ((¬P¦(a1)+(P¦(x1)+P¦(c1)))&(¬P¦(a1)+(¬P¦(b1)+P¦(c1))))
29 | | | | *show (¬P¦(a1)+(P¦(x1)+P¦(c1)))
30 | | | | | ¬(¬P¦(a1)+(P¦(x1)+P¦(c1)))                                                                           ASSUME
31 | | | | | ((P¦(a1)&(P¦(x1)→P¦(b1)))→P¦(c1))                                                                    26,UI
32 | | | | | ((P¦(a1)&(P¦(a1)→P¦(b1)))→P¦(c1))                                                                    26,UI
33 | | | | | ((P¦(a1)&(P¦(c1)→P¦(b1)))→P¦(c1))                                                                    26,UI
34 | | | | | ((P¦(a1)&(P¦(b1)→P¦(b1)))→P¦(c1))                                                                    26,UI
35 | | | | | *show ¬P¦(a1)
36 | | | | | | P¦(a1)                                                                                             ASSUME
37 | | | | | | *show (P¦(x1)+P¦(c1))
38 | | | | | | | ¬(P¦(x1)+P¦(c1))                                                                                 ASSUME
39 | | | | | | | *show P¦(x1)
40 | | | | | | | | ¬P¦(x1)                                                                                          ASSUME
41 | | | | | | | | *show P¦(c1)
42 | | | | | | | | | ¬P¦(c1)                                                                                        ASSUME
43 | | | | | | | | | ¬(P¦(a1)&(P¦(x1)→P¦(b1)))                                                                      42,31,MT
44 | | | | | | | | | ¬(P¦(a1)&(P¦(a1)→P¦(b1)))                                                                      42,32,MT
45 | | | | | | | | | ¬(P¦(a1)&(P¦(c1)→P¦(b1)))                                                                      42,33,MT
46 | | | | | | | | | ¬(P¦(a1)&(P¦(b1)→P¦(b1)))                                                                      42,34,MT
47 | | | | | | | | | *show (P¦(a1)&(P¦(x1)→P¦(b1)))
```

```
48 | | | | | | | | | | P¦(a₁)                                                                                    36,R
49 | | | | | | | | | | *show (P¦(x₁)→P¦(b₁))
50 | | | | | | | | | | | P¦(x₁)                                                                                  ASSUME
51 | | | | | | | | | | | ¬P¦(x₁)                                                                                 40,R
52 | | | | | | | | | | (P¦(a₁)&(P¦(x₁)→P¦(b₁)))                                                                  48,49,ADJ
53 | | | | | | | | | ¬(P¦(x₁)+P¦(c₁))                                                                          38,R
54 | | | | | | | | | (P¦(x₁)+P¦(c₁))                                                                           41,ADD
55 | | | | | | | | (P¦(x₁)+P¦(c₁))                                                                           39,ADD
56 | | | | | | | ¬(¬P¦(a₁)+(P¦(x₁)+P¦(c₁)))                                                                30,R
57 | | | | | | | (¬P¦(a₁)+(P¦(x₁)+P¦(c₁)))                                                                 37,ADD
58 | | | | | | (¬P¦(a₁)+(P¦(x₁)+P¦(c₁)))                                                                 35,ADD
59 | | | | | *show (¬P¦(a₁)+(¬P¦(b₁)+P¦(c₁)))
60 | | | | | | ¬(¬P¦(a₁)+(¬P¦(b₁)+P¦(c₁)))                                                               ASSUME
61 | | | | | | ((P¦(a₁)&(P¦(x₁)→P¦(b₁)))+P¦(c₁))                                                         26,UI
62 | | | | | | ((P¦(a₁)&(P¦(a₁)→P¦(b₁)))+P¦(c₁))                                                         26,UI
63 | | | | | | ((P¦(a₁)&(P¦(c₁)→P¦(b₁)))+P¦(c₁))                                                         26,UI
64 | | | | | | ((P¦(a₁)&(P¦(b₁)→P¦(b₁)))+P¦(c₁))                                                         26,UI
65 | | | | | | *show ¬P¦(a₁)
66 | | | | | | | P¦(a₁)                                                                                  ASSUME
67 | | | | | | | (P¦(x₁)+P¦(c₁))                                                                         66,29,MTP
68 | | | | | | | *show (¬P¦(b₁)+P¦(c₁))
69 | | | | | | | | ¬(¬P¦(b₁)+P¦(c₁))                                                                     ASSUME
70 | | | | | | | | *show ¬P¦(b₁)
71 | | | | | | | | | P¦(b₁)                                                                                ASSUME
72 | | | | | | | | | *show P¦(c₁)
73 | | | | | | | | | | ¬P¦(c₁)                                                                             ASSUME
74 | | | | | | | | | | ¬(P¦(a₁)&(P¦(x₁)+P¦(b₁)))                                                           73,61,MT
75 | | | | | | | | | | ¬(P¦(a₁)&(P¦(a₁)+P¦(b₁)))                                                           73,62,MT
76 | | | | | | | | | | ¬(P¦(a₁)&(P¦(c₁)+P¦(b₁)))                                                           73,63,MT
77 | | | | | | | | | | ¬(P¦(a₁)&(P¦(b₁)+P¦(b₁)))                                                           73,64,MT
78 | | | | | | | | | | P¦(x₁)                                                                              73,67,MTP
79 | | | | | | | | | | *show (P¦(a₁)&(P¦(x₁)→P¦(b₁)))
80 | | | | | | | | | | | P¦(a₁)                                                                            66,R
81 | | | | | | | | | | | *show (P¦(x₁)→P¦(b₁))
82 | | | | | | | | | | | | P¦(b₁)                                                                          71,R
83 | | | | | | | | | | | (P¦(a₁)&(P¦(x₁)→P¦(b₁)))                                                          80,81,ADJ
84 | | | | | | | | | ¬(¬P¦(b₁)+P¦(c₁))                                                                   69,R
85 | | | | | | | | | (¬P¦(b₁)+P¦(c₁))                                                                    72,ADD
86 | | | | | | | | (¬P¦(b₁)+P¦(c₁))                                                                    70,ADD
87 | | | | | | | ¬(¬P¦(a₁)+(¬P¦(b₁)+P¦(c₁)))                                                         60,R
88 | | | | | | | (¬P¦(a₁)+(¬P¦(b₁)+P¦(c₁)))                                                          68,ADD
89 | | | | | | (¬P¦(a₁)+(¬P¦(b₁)+P¦(c₁)))                                                          65,ADD
90 | | | | | ((¬P¦(a₁)+(P¦(x₁)+P¦(c₁)))&(¬P¦(a₁)+(¬P¦(b₁)+P¦(c₁))))                                      29,59,ADJ
91 | ((Ax₁)((P¦(a₁)&(P¦(x₁)→P¦(b₁)))+P¦(c₁))↔(Ax₁)((¬P¦(a₁)+(P¦(x₁)+P¦(c₁)))&(¬P¦(a₁)+(¬P¦(b₁)+P¦(c₁))))) 25,2,CB
```

CPU time: 701 msec
Statements: 45165

```
1  *show ((Ax1)((P1(a1)&(P1(x1)→(Ey1)(P1(y1)&R21(x1,y1))))→(Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1))))↔
   |    (Ax1)((¬P1(a1)+(P1(x1)+(Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1)))))&(¬P1(a1)+(¬(Ey1)(P1(y1)&R21(x1,y1))+
   |    (Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1))))))
2  |  *show ((Ax1)((¬P1(a1)+(P1(x1)+(Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1)))))&(¬P1(a1)+(¬(Ey1)(P1(y1)&R21(x1,y1))+
   |    |    (Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1))))))→(Ax1)((P1(a1)&(P1(x1)→(Ey1)(P1(y1)&R21(x1,y1))))+
   |    |    (Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1)))))
   |    (Ax1)((¬P1(a1)+(P1(x1)+(Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1)))))&(¬P1(a1)+(¬(Ey1)(P1(y1)&R21(x1,y1))+
3  |    (Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1))))))                                                    ASSUME
4  |    *show (Ax1)((P1(a1)&(P1(x1)→(Ey1)(P1(y1)&R21(x1,y1))))→(Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1))))
5  |    |  *show ((P1(a1)&(P1(x1)→(Ey1)(P1(y1)&R21(x1,y1))))→(Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1))))
6  |    |  |  (P1(a1)&(P1(x1)→(Ey1)(P1(y1)&R21(x1,y1))))                                                ASSUME
7  |    |  |  *show (Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1)))
8  |    |  |  |  ¬(Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1)))                                             ASSUME
9  |    |  |  |  P1(a1)                                                                                  6,S
10 |    |  |  |  (P1(x1)→(Ey1)(P1(y1)&R21(x1,y1)))                                                       6,S
11 |    |  |  |  (Az1)¬(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1)))                                             8,QN
   |    |  |  |  ((¬P1(a1)+(P1(x1)+(Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1)))))&(¬P1(a1)+(¬(Ey1)(P1(y1)&R21(x1,y1))+
12 |    |  |  |    (Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1))))))                                          3,UI
13 |    |  |  |  ¬(Ew1)(P1(x1)&(R21(x1,w1)&R21(w1,x1)))                                                  11,UI
   |    |  |  |  ((¬P1(a1)+(P1(a1)+(Ez1)(Ew1)(P1(z1)&(R21(a1,w1)&R21(w1,z1)))))&(¬P1(a1)+(¬(Ey1)(P1(y1)&R21(a1,y1))+
14 |    |  |  |    (Ez1)(Ew1)(P1(z1)&(R21(a1,w1)&R21(w1,z1))))))                                          3,UI
15 |    |  |  |  ¬(Ew1)(P1(a1)&(R21(x1,w1)&R21(w1,a1)))                                                  11,UI
16 |    |  |  |  (¬P1(a1)+(P1(x1)+(Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1)))))                           12,S
17 |    |  |  |  (¬P1(a1)+(¬(Ey1)(P1(y1)&R21(x1,y1))+(Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1)))))  12,S
18 |    |  |  |  (¬P1(a1)+(P1(a1)+(Ez1)(Ew1)(P1(z1)&(R21(a1,w1)&R21(w1,z1)))))                           14,S
19 |    |  |  |  (¬P1(a1)+(¬(Ey1)(P1(y1)&R21(a1,y1))+(Ez1)(Ew1)(P1(z1)&(R21(a1,w1)&R21(w1,z1)))))  14,S
20 |    |  |  |  (P1(x1)+(Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1))))                                     9,16,MTP
21 |    |  |  |  (¬(Ey1)(P1(y1)&R21(x1,y1))+(Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1))))                  9,17,MTP
22 |    |  |  |  (P1(a1)+(Ez1)(Ew1)(P1(z1)&(R21(a1,w1)&R21(w1,z1))))                                     9,18,MTP
23 |    |  |  |  (¬(Ey1)(P1(y1)&R21(a1,y1))+(Ez1)(Ew1)(P1(z1)&(R21(a1,w1)&R21(w1,z1))))                  9,19,MTP
24 |    |  |  |  P1(x1)                                                                                  8,20,MTP
25 |    |  |  |  ¬(Ey1)(P1(y1)&R21(x1,y1))                                                               8,21,MTP
26 |    |  |  |  (Aw1)¬(P1(x1)&(R21(x1,w1)&R21(w1,x1)))                                                  13,QN
27 |    |  |  |  (Aw1)¬(P1(a1)&(R21(x1,w1)&R21(w1,a1)))                                                  15,QN
28 |    |  |  |  (Ay1)¬(P1(y1)&R21(x1,y1))                                                               25,QN
29 |    |  |  |  ¬(P1(x1)&(R21(x1,x1)&R21(x1,x1)))                                                       26,UI
30 |    |  |  |  ¬(P1(a1)&(R21(x1,x1)&R21(x1,a1)))                                                       27,UI
31 |    |  |  |  ¬(P1(x1)&R21(x1,x1))                                                                    28,UI
32 |    |  |  |  ¬(P1(x1)&(R21(x1,a1)&R21(a1,x1)))                                                       26,UI
33 |    |  |  |  ¬(P1(a1)&(R21(x1,a1)&R21(a1,a1)))                                                       27,UI
34 |    |  |  |  ¬(P1(a1)&R21(x1,a1))                                                                    28,UI
35 |    |  |  |  (Ey1)(P1(y1)&R21(x1,y1))                                                                10,24,MP
36 |    *show ((Ax1)((P1(a1)&(P1(x1)→(Ey1)(P1(y1)&R21(x1,y1))))→(Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1))))+
   |    |    (Ax1)((¬P1(a1)+(P1(x1)+(Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1)))))&(¬P1(a1)+(¬(Ey1)(P1(y1)&R21(x1,y1))+
   |    |    (Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1))))))
37 |    |  (Ax1)((P1(a1)&(P1(x1)→(Ey1)(P1(y1)&R21(x1,y1))))→(Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1)))) ASSUME
38 |    |  *show (Ax1)((¬P1(a1)+(P1(x1)+(Ez1)(Ew1)(P1(z1)&(R21(x1,w1)&R21(w1,z1)))))&(¬P1(a1)+(¬(Ey1)(P1(y1)&R21(x1,y1))+
```

```
    (Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1))))))
39  *show ((¬P¦1(a1)+(P¦1(x1)+(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1)))))&(¬P¦1(a1)+(¬(Ey1)(P¦1(y1)&R²1(x1,y1))+
        (Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1))))))
40    *show (¬P¦1(a1)+(P¦1(x1)+(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1)))))
41      ¬(¬P¦1(a1)+(P¦1(x1)+(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1))))) ASSUME
42      ((P¦1(a1)&(P¦1(x1)→(Ey1)(P¦1(y1)&R²1(x1,y1))))→(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1)))) 37,UI
43      ((P¦1(a1)&(P¦1(a1)→(Ey1)(P¦1(y1)&R²1(a1,y1))))→(Ez1)(Ew1)(P¦1(z1)&(R²1(a1,w1)&R²1(w1,z1)))) 37,UI
44      *show ¬P¦1(a1)
45        P¦1(a1)                                                          ASSUME
46        *show (P¦1(x1)+(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1))))
47          ¬(P¦1(x1)+(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1))))       ASSUME
48          *show P¦1(x1)
49            ¬P¦1(x1)                                                   ASSUME
50            *show (Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1)))
51              ¬(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1)))             ASSUME
52              ¬(P¦1(a1)&(P¦1(x1)→(Ey1)(P¦1(y1)&R²1(x1,y1))))           51,42,MT
53              (Az1)¬(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1)))             51,QN
54              ¬(Ew1)(P¦1(x1)&(R²1(x1,w1)&R²1(w1,x1)))                  53,UI
55              ¬(Ew1)(P¦1(a1)&(R²1(x1,w1)&R²1(w1,a1)))                  53,UI
56              (Aw1)¬(P¦1(x1)&(R²1(x1,w1)&R²1(w1,x1)))                  54,QN
57              (Aw1)¬(P¦1(a1)&(R²1(x1,w1)&R²1(w1,a1)))                  55,QN
58              ¬(P¦1(x1)&(R²1(x1,x1)&R²1(x1,x1)))                       56,UI
59              ¬(P¦1(a1)&(R²1(x1,x1)&R²1(x1,a1)))                       57,UI
60              ¬(P¦1(x1)&(R²1(x1,a1)&R²1(a1,x1)))                       56,UI
61              ¬(P¦1(a1)&(R²1(x1,a1)&R²1(a1,a1)))                       57,UI
62              *show (P¦1(a1)&(P¦1(x1)→(Ey1)(P¦1(y1)&R²1(x1,y1))))
63                P¦1(a1)
                                                                         45,R
64                *show (P¦1(x1)→(Ey1)(P¦1(y1)&R²1(x1,y1)))
65                  P¦1(x1)
                                                                         ASSUME
66                  ¬P¦1(x1)
                                                                         49,R
67                (P¦1(a1)&(P¦1(x1)→(Ey1)(P¦1(y1)&R²1(x1,y1))))          63,64,ADJ
68            ¬(P¦1(x1)+(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1))))     47,R
69            (P¦1(x1)+(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1))))      50,ADD
70          (P¦1(x1)+(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1))))        48,ADD
71        ¬(¬P¦1(a1)+(P¦1(x1)+(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1)))))  41,R
72        (¬P¦1(a1)+(P¦1(x1)+(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1)))))  46,ADD
73      (¬P¦1(a1)+(P¦1(x1)+(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1)))))  44,ADD
74  *show (¬P¦1(a1)+(¬(Ey1)(P¦1(y1)&R²1(x1,y1))+(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1)))))
75    ¬(¬P¦1(a1)+(¬(Ey1)(P¦1(y1)&R²1(x1,y1))+(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1))))) ASSUME
76    ((P¦1(a1)&(P¦1(x1)→(Ey1)(P¦1(y1)&R²1(x1,y1))))→(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1)))) 37,UI
77    ((P¦1(a1)&(P¦1(a1)→(Ey1)(P¦1(y1)&R²1(a1,y1))))→(Ez1)(Ew1)(P¦1(z1)&(R²1(a1,w1)&R²1(w1,z1)))) 37,UI
78    *show ¬P¦1(a1)
79      P¦1(a1)                                                            ASSUME
80      (P¦1(x1)+(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1))))              79,40,MTP
81      *show (¬(Ey1)(P¦1(y1)&R²1(x1,y1))+(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1))))
82        ¬(¬(Ey1)(P¦1(y1)&R²1(x1,y1))+(Ez1)(Ew1)(P¦1(z1)&(R²1(x1,w1)&R²1(w1,z1)))) ASSUME
83        *show ¬(Ey1)(P¦1(y1)&R²1(x1,y1))
```

```
84  | | | | | | | | (Ey1)(P1(y1)&R2(x1,y1))                                                
```

Next follows proofs relevant to the discussions in Chapter VII on the EI/UI problem.

29. $\vdash (Ey)(Ax)(Py \rightarrow Px)$

30. $\vdash (Ex)(Ey)(Pxy \rightarrow (Ax)(Ay)Pxy)$

31. $\vdash (Ex)(Ay)(Az)((Py \rightarrow Qz) \rightarrow (Px \rightarrow Qx))$

32. $\vdash ((Ax)(Ay)(Ez)(Aw)((Px \& Qy) \rightarrow (Rz \& Sw)) \rightarrow$
$((Ex)(Ey)(Px \& Qy) \rightarrow (Ez)Rz))$

```
 1 *show (Ey,)(Ax,)(P¦(y,)→P¦(x,))
 2 | ¬(Ey,)(Ax,)(P¦(y,)→P¦(x,))                    ASSUME
 3 | (Ay,)¬(Ax,)(P¦(y,)→P¦(x,))                   2,QN
 4 | ¬(Ax,)(P¦(z.)→P¦(x,))                         3,UI
 5 | (Ex,)¬(P¦(z.)→P¦(x,))                         4,QN
 6 | ¬(P¦(z.)→P¦(z.))                              5,EI
 7 | *show (P¦(z.)→P¦(z.))
 8 | | P¦(z.)                                      ASSUME
 9 | | *show P¦(z.)
10 | | | ¬P¦(z.)                                   ASSUME
11 | | | ¬(Ax,)(P¦(z.)→P¦(x,))                     3,UI
12 | | | (Ex,)¬(P¦(z.)→P¦(x,))                     11,QN
13 | | | ¬(P¦(z.)→P¦(z,))                          12,EI
14 | | | *show (P¦(z.)→P¦(z,))
15 | | | | P¦(z.)                                  ASSUME
16 | | | | ¬P¦(z.)                                 10,R
```

CPU time: 112 msec
Statements: 7586

```
1  *show (Ex1)(Ey1)(P2(x1,y1)→(Ax1)(Ay1)P2(x1,y1))
2  | ¬(Ex1)(Ey1)(P2(x1,y1)→(Ax1)(Ay1)P2(x1,y1))                    ASSUME
3  | (Ax1)¬(Ey1)(P2(x1,y1)→(Ax1)(Ay1)P2(x1,y1))                    2,QN
4  | ¬(Ey1)(P2(z0,y1)→(Ax1)(Ay1)P2(x1,y1))                         3,UI
5  | (Ay1)¬(P2(z0,y1)→(Ax1)(Ay1)P2(x1,y1))                         4,QN
6  | ¬(P2(z0,z0)→(Ax1)(Ay1)P2(x1,y1))                              5,UI
7  | *show (P2(z0,z0)→(Ax1)(Ay1)P2(x1,y1))
8  | | P2(z0,z0)                                                   ASSUME
9  | | *show (Ax1)(Ay1)P2(x1,y1)
10 | | | *show (Ay1)P2(x1,y1)
11 | | | | *show P2(x1,y1)
12 | | | | ¬P2(x1,y1)                                              ASSUME
13 | | | | ¬(Ey1)(P2(x1,y1)→(Ax1)(Ay1)P2(x1,y1))                   3,UI
14 | | | | ¬(Ey1)(P2(y1,y1)→(Ax1)(Ay1)P2(x1,y1))                   3,UI
15 | | | | ¬(P2(z0,x1)→(Ax1)(Ay1)P2(x1,y1))                        5,UI
16 | | | | ¬(P2(z0,y1)→(Ax1)(Ay1)P2(x1,y1))                        5,UI
17 | | | | (Ay1)¬(P2(x1,y1)→(Ax1)(Ay1)P2(x1,y1))                   13,QN
18 | | | | (Ay1)¬(P2(y1,y1)→(Ax1)(Ay1)P2(x1,y1))                   14,QN
19 | | | | ¬(P2(x1,z0)→(Ax1)(Ay1)P2(x1,y1))                        17,UI
20 | | | | ¬(P2(x1,x1)→(Ax1)(Ay1)P2(x1,y1))                        17,UI
21 | | | | ¬(P2(x1,y1)→(Ax1)(Ay1)P2(x1,y1))                        17,UI
22 | | | | ¬(P2(y1,y1)→(Ax1)(Ay1)P2(x1,y1))                        18,UI
23 | | | | *show (P2(z0,x1)→(Ax1)(Ay1)P2(x1,y1))
24 | | | | | P2(z0,x1)                                             ASSUME
25 | | | | | *show (P2(z0,y1)→(Ax1)(Ay1)P2(x1,y1))
26 | | | | | | P2(z0,y1)                                           ASSUME
27 | | | | | | *show (P2(x1,z0)→(Ax1)(Ay1)P2(x1,y1))
28 | | | | | | | P2(x1,z0)                                         ASSUME
29 | | | | | | | *show (P2(x1,x1)→(Ax1)(Ay1)P2(x1,y1))
30 | | | | | | | | P2(x1,x1)                                       ASSUME
31 | | | | | | | | *show (P2(x1,y1)→(Ax1)(Ay1)P2(x1,y1))
32 | | | | | | | | | P2(x1,y1)                                     ASSUME
33 | | | | | | | | | ¬P2(x1,y1)                                    12,R
34 | | | | | | | | ¬(P2(x1,y1)→(Ax1)(Ay1)P2(x1,y1))                21,R
35 | | | | | | | ¬(P2(x1,x1)→(Ax1)(Ay1)P2(x1,y1))                  20,R
36 | | | | | | ¬(P2(x1,z0)→(Ax1)(Ay1)P2(x1,y1))                    19,R
37 | | | | | ¬(P2(z0,y1)→(Ax1)(Ay1)P2(x1,y1))                      16,R
```

CPU time: 304 msec
Statements: 23217

```
1  *show (Ex1)(Ay1)(Az1)((P¦(y1)→Q¦(z1))→(P¦(x1)→Q¦(x1)))
2  | ¬(Ex1)(Ay1)(Az1)((P¦(y1)→Q¦(z1))→(P¦(x1)→Q¦(x1)))                    ASSUME
3  | (Ax1)¬(Ay1)(Az1)((P¦(y1)→Q¦(z1))→(P¦(x1)→Q¦(x1)))                    2,QN
4  | ¬(Ay1)(Az1)((P¦(y1)→Q¦(z1))→(P¦(z2)→Q¦(z2)))                        3,UI
5  | (Ey1)¬(Az1)((P¦(y1)→Q¦(z1))→(P¦(z2)→Q¦(z2)))                        4,QN.
6  | ¬(Az1)((P¦(z2)→Q¦(z1))→(P¦(z2)→Q¦(z2)))                            5,EI
7  | (Ez1)¬((P¦(z2)→Q¦(z1))→(P¦(z2)→Q¦(z2)))                            6,QN
8  | ¬((P¦(z2)→Q¦(z1))→(P¦(z2)→Q¦(z2)))                                 7,EI
9  | *show ((P¦(z2)→Q¦(z1))→(P¦(z2)→Q¦(z2)))
10 | | (P¦(z2)→Q¦(z1))                                                  ASSUME
11 | | *show (P¦(z2)→Q¦(z2))
12 | | | P¦(z2)                                                        ASSUME
13 | | | *show Q¦(z2)
14 | | | | ¬Q¦(z2)                                                     ASSUME
15 | | | | *show P¦(z2)
16 | | | | | ¬P¦(z2)                                                   ASSUME
17 | | | | | ¬(Ay1)(Az1)((P¦(y1)→Q¦(z1))→(P¦(z2)→Q¦(z2)))              3,UI
18 | | | | | ¬(Ay1)(Az1)((P¦(y1)→Q¦(z1))→(P¦(z1)→Q¦(z1)))              3,UI
19 | | | | | (Ey1)¬(Az1)((P¦(y1)→Q¦(z1))→(P¦(z2)→Q¦(z2)))              17,QN
20 | | | | | (Ey1)¬(Az1)((P¦(y1)→Q¦(z1))→(P¦(z1)→Q¦(z1)))              18,QN
21 | | | | | ¬(Az1)((P¦(z2)→Q¦(z1))→(P¦(z2)→Q¦(z2)))                   19,EI
22 | | | | | ¬(Az1)((P¦(z2)→Q¦(z1))→(P¦(z1)→Q¦(z1)))                   20,EI
23 | | | | | (Ez1)¬((P¦(z2)→Q¦(z1))→(P¦(z2)→Q¦(z2)))                   21,QN
24 | | | | | (Ez1)¬((P¦(z2)→Q¦(z1))→(P¦(z1)→Q¦(z1)))                   22,QN
25 | | | | | ¬((P¦(z2)→Q¦(z2))→(P¦(z2)→Q¦(z2)))                        23,EI
26 | | | | | ¬((P¦(z2)→Q¦(z1))→(P¦(z1)→Q¦(z1)))                        24,EI
27 | | | | | *show ((P¦(z2)→Q¦(z2))→(P¦(z2)→Q¦(z2)))
28 | | | | | | (P¦(z2)→Q¦(z2))                                       ASSUME
29 | | | | | | *show (P¦(z2)→Q¦(z2))
30 | | | | | | | P¦(z2)                                              ASSUME
31 | | | | | | | ¬P¦(z2)                                             16,R
32 | | | | | Q¦(z2)                                                    10,15,MP
33 | | | | | ¬(Ay1)(Az1)((P¦(y1)→Q¦(z1))→(P¦(z2)→Q¦(z2)))              3,UI
34 | | | | | ¬(Ay1)(Az1)((P¦(y1)→Q¦(z1))→(P¦(z1)→Q¦(z1)))              3,UI
35 | | | | | ¬(Ay1)(Az1)((P¦(y1)→Q¦(z1))→(P¦(z2)→Q¦(z2)))              3,UI
36 | | | | | ¬(Ay1)(Az1)((P¦(y1)→Q¦(z1))→(P¦(z2)→Q¦(z2)))              3,UI
37 | | | | | ¬(Ay1)(Az1)((P¦(y1)→Q¦(z1))→(P¦(z2)→Q¦(z2)))              3,UI
38 | | | | | ¬(Ay1)(Az1)((P¦(y1)→Q¦(z1))→(P¦(z1)→Q¦(z1)))              3,UI
39 | | | | | (Ey1)¬(Az1)((P¦(y1)→Q¦(z1))→(P¦(z2)→Q¦(z2)))              33,QN
40 | | | | | (Ey1)¬(Az1)((P¦(y1)→Q¦(z1))→(P¦(z1)→Q¦(z1)))              34,QN
41 | | | | | (Ey1)¬(Az1)((P¦(y1)→Q¦(z1))→(P¦(z2)→Q¦(z2)))              35,QN
42 | | | | | (Ey1)¬(Az1)((P¦(y1)→Q¦(z1))→(P¦(z1)→Q¦(z1)))              36,QN
43 | | | | | (Ey1)¬(Az1)((P¦(y1)→Q¦(z1))→(P¦(z2)→Q¦(z2)))              37,QN
44 | | | | | (Ey1)¬(Az1)((P¦(y1)→Q¦(z1))→(P¦(z1)→Q¦(z1)))              38,QN
45 | | | | | ¬(Az1)((P¦(z2)→Q¦(z1))→(P¦(z2)→Q¦(z2)))                   39,EI
46 | | | | | ¬(Az1)((P¦(z2)→Q¦(z1))→(P¦(z1)→Q¦(z1)))                   40,EI
47 | | | | | ¬(Az1)((P¦(y1)→Q¦(z1))→(P¦(z2)→Q¦(z2)))                   41,EI
```

```
48 | | | | ¬(Az₁)((P¦(y₀)→Q¦(z₁))→(P¦(z₀)→Q¦(z₀)))                    42,EI
49 | | | | ¬(Az₁)((P¦(y₁)→Q¦(z₁))→(P¦(z₀)→Q¦(z₀)))                    43,EI
50 | | | | ¬(Az₁)((P¦(y₀)→Q¦(z₁))→(P¦(z₁)→Q¦(z₁)))                    44,EI
51 | | | | (Ez₁)¬((P¦(z₁)→Q¦(z₁))→(P¦(z₀)→Q¦(z₀)))                    45,QN
52 | | | | (Ez₁)¬((P¦(z₀)→Q¦(z₁))→(P¦(z₁)→Q¦(z₁)))                    46,QN
53 | | | | (Ez₁)¬((P¦(y₀)→Q¦(z₁))→(P¦(z₀)→Q¦(z₀)))                    47,QN
54 | | | | (Ez₁)¬((P¦(y₀)→Q¦(z₁))→(P¦(z₁)→Q¦(z₀)))                    48,QN
55 | | | | (Ez₁)¬((P¦(y₁)→Q¦(z₁))→(P¦(z₀)→Q¦(z₀)))                    49,QN
56 | | | | (Ez₁)¬((P¦(y₀)→Q¦(z₁))→(P¦(z₁)→Q¦(z₁)))                    50,QN
57 | | | | ¬((P¦(z₁)→Q¦(y₀))→(P¦(z₀)→Q¦(z₀)))                         51,EI
58 | | | | ¬((P¦(z₀)→Q¦(y₀))→(P¦(z₁)→Q¦(z₁)))                         52,EI
59 | | | | ¬((P¦(y₀)→Q¦(y₁))→(P¦(z₀)→Q¦(z₀)))                         53,EI
60 | | | | ¬((P¦(y₀)→Q¦(y₁))→(P¦(z₁)→Q¦(z₀)))                         54,EI
61 | | | | ¬((P¦(y₁)→Q¦(y₀))→(P¦(z₁)→Q¦(z₀)))                         55,EI
62 | | | | ¬((P¦(y₀)→Q¦(x₀))→(P¦(z₁)→Q¦(z₁)))                         56,EI
63 | | | | *show ((P¦(z₁)→Q¦(y₀))→(P¦(z₀)→Q¦(z₀)))
64 | | | |  (P¦(z₁)→Q¦(y₀))                                          ASSUME
65 | | | |  *show (P¦(z₀)→Q¦(z₀))
66 | | | |  | P¦(z₀)                                                 ASSUME
67 | | | |  | *show Q¦(z₀)
68 | | | |  | | ¬Q¦(z₀)                                            ASSUME
69 | | | |  | | *show ((P¦(z₀)→Q¦(y₀))→(P¦(z₁)→Q¦(z₁)))
70 | | | |  | | | (P¦(z₀)→Q¦(y₀))                                  ASSUME
71 | | | |  | | | *show (P¦(z₁)→Q¦(z₁))
72 | | | |  | | | | Q¦(z₁)                                         32,R
73 | | | |  | | | ¬((P¦(z₀)→Q¦(y₀))→(P¦(z₁)→Q¦(z₁)))                58,R
```

CPU time: 570 msec
Statements: 41333

```
1  *show ((Ax1)(Ay1)(Ez1)(Aw1)((P¦(x1)&Q¦(y1))→(R¦(z1)&S¦(w1)))→((Ex1)(Ey1)(P¦(x1)&Q¦(y1))→(Ez1)R¦(z1)))
2  | (Ax1)(Ay1)(Ez1)(Aw1)((P¦(x1)&Q¦(y1))→(R¦(z1)&S¦(w1)))                                    ASSUME
3  | *show ((Ex1)(Ey1)(P¦(x1)&Q¦(y1))→(Ez1)R¦(z1))
4  | | (Ex1)(Ey1)(P¦(x1)&Q¦(y1))                                                              ASSUME
5  | | *show (Ez1)R¦(z1)
6  | | | ¬(Ez1)R¦(z1)                                                                         ASSUME
7  | | | (Ey1)(P¦(z2)&Q¦(y1))                                                                 4,EI
8  | | | (P¦(z2)&Q¦(z2))                                                                      7,EI
9  | | | (Az1)¬R¦(z1)                                                                         6,QN
10 | | | (Ay1)(Ez1)(Aw1)((P¦(z2)&Q¦(y1))→(R¦(z1)&S¦(w1)))                                     2,UI
11 | | | (Ay1)(Ez1)(Aw1)((P¦(z2)&Q¦(y1))→(R¦(z1)&S¦(w1)))                                     2,UI
12 | | | ¬R¦(z2)                                                                              9,UI
13 | | | ¬R¦(z2)                                                                              9,UI
14 | | | (Ez1)(Aw1)((P¦(z2)&Q¦(z2))→(R¦(z1)&S¦(w1)))                                          10,UI
15 | | | (Ez1)(Aw1)((P¦(z2)&Q¦(z2))→(R¦(z1)&S¦(w1)))                                          10,UI
16 | | | (Ez1)(Aw1)((P¦(z2)&Q¦(z2))→(R¦(z1)&S¦(w1)))                                          11,UI
17 | | | (Ez1)(Aw1)((P¦(z2)&Q¦(z2))→(R¦(z1)&S¦(w1)))                                          11,UI
18 | | | P¦(z2)                                                                               8,S
19 | | | Q¦(z2)                                                                               8,S
20 | | | (Aw1)((P¦(z2)&Q¦(z2))→(R¦(z1)&S¦(w1)))                                               14,EI
21 | | | (Aw1)((P¦(z2)&Q¦(z2))→(R¦(z2)&S¦(w1)))                                               15,EI
22 | | | (Aw1)((P¦(z2)&Q¦(z2))→(R¦(z2)&S¦(w1)))                                               16,EI
23 | | | (Aw1)((P¦(z2)&Q¦(z2))→(R¦(z2)&S¦(w1)))                                               17,EI
24 | | | ((P¦(z2)&Q¦(z2))→(R¦(z1)&S¦(z2)))                                                    20,UI
25 | | | ((P¦(z2)&Q¦(z2))→(R¦(z2)&S¦(z2)))                                                    20,UI
26 | | | ((P¦(z2)&Q¦(z2))→(R¦(z2)&S¦(z2)))                                                    21,UI
27 | | | ((P¦(z2)&Q¦(z2))→(R¦(z2)&S¦(z2)))                                                    21,UI
28 | | | ((P¦(z2)&Q¦(z2))→(R¦(z2)&S¦(z2)))                                                    22,UI
29 | | | ((P¦(z2)&Q¦(z2))→(R¦(z2)&S¦(z2)))                                                    22,UI
30 | | | ((P¦(z2)&Q¦(z2))→(R¦(z2)&S¦(z2)))                                                    23,UI
31 | | | ((P¦(z2)&Q¦(z2))→(R¦(z2)&S¦(z2)))                                                    23,UI
32 | | | (R¦(z2)&S¦(z2))                                                                      26,8,MP
33 | | | (R¦(z2)&S¦(z2))                                                                      27,8,MP
34 | | | R¦(z2)                                                                               32,S
35 | | | (Ez1)R¦(z1)                                                                          34,EG
```

CPU time: 361 msec
Statements: 25433

Finally we have the six subproblems of Andrew's Challenge, which was fully discussed in Chapter VII, Section J. (The six subproblems are what are on line 1 of the following six proofs).

```
1  *show (((Ex1)(Ay1)(P¦(x1)↔P¦(y1))↔((Ex2)Q¦(x2)↔(Ay2)P¦(y2)))→((Ex3)(Ay3)(Q¦(x3)↔Q¦(y3))→((Ex4)P¦(x4)→(Ay4)Q¦(y4))))
2  | ((Ex1)(Ay1)(P¦(x1)↔P¦(y1))↔((Ex2)Q¦(x2)↔(Ay2)P¦(y2)))                                                    ASSUME
3  | *show ((Ex3)(Ay3)(Q¦(x3)↔Q¦(y3))→((Ex4)P¦(x4)→(Ay4)Q¦(y4)))
4  | | (Ex3)(Ay3)(Q¦(x3)↔Q¦(y3))                                                                             ASSUME
5  | | *show ((Ex4)P¦(x4)→(Ay4)Q¦(y4))
6  | | | (Ex4)P¦(x4)                                                                                         ASSUME
7  | | | *show (Ay4)Q¦(y4)
8  | | | | *show Q¦(y4)
9  | | | | | ¬Q¦(y4)                                                                                           ASSUME
10 | | | | | ((Ex1)(Ay1)(P¦(x1)↔P¦(y1))→((Ex2)Q¦(x2)↔(Ay2)P¦(y2)))                                              2,BC
11 | | | | | (((Ex2)Q¦(x2)↔(Ay2)P¦(y2))→(Ex3)(Ay3)(P¦(x3)↔P¦(y3)))                                              2,BC
12 | | | | | (Ay3)(Q¦(z4)↔Q¦(y3))                                                                              4,EI
13 | | | | | P¦(z4)                                                                                            6,EI
14 | | | | | (Q¦(z4)↔Q¦(y4))                                                                                   12,UI
15 | | | | | (Q¦(z1)↔Q¦(z1))                                                                                   12,UI
16 | | | | | (Q¦(z4)↔Q¦(z4))                                                                                   12,UI
17 | | | | | (Q¦(z4)→Q¦(y4))                                                                                   14,BC
18 | | | | | (Q¦(y4)→Q¦(z4))                                                                                   14,BC
19 | | | | | (Q¦(z1)→Q¦(z1))                                                                                   15,BC
20 | | | | | (Q¦(z4)→Q¦(z4))                                                                                   16,BC
21 | | | | | (Q¦(z4)→Q¦(z4))                                                                                   16,BC
22 | | | | | ¬Q¦(z4)                                                                                           9,17,MT
23 | | | | | ¬Q¦(z4)                                                                                           22,21,MT
24 | | | | | *show (Ex1)(Ay1)(P¦(x1)↔P¦(y1))
25 | | | | | | ¬(Ex1)(Ay1)(P¦(x1)↔P¦(y1))                                                                      ASSUME
26 | | | | | | ¬((Ex2)Q¦(x2)↔(Ay2)P¦(y2))                                                                      25,11,MT
27 | | | | | | (Ax1)¬(Ay1)(P¦(x1)↔P¦(y1))                                                                      25,QN
28 | | | | | | ¬(Ay1)(P¦(y4)↔P¦(y1))                                                                           27,UI
29 | | | | | | ¬(Ay1)(P¦(z4)↔P¦(y1))                                                                           27,UI
30 | | | | | | ¬(Ay1)(P¦(z4)↔P¦(y1))                                                                           27,UI
31 | | | | | | (Ey1)¬(P¦(y4)↔P¦(y1))                                                                           28,QN
32 | | | | | | (Ey1)¬(P¦(z4)↔P¦(y1))                                                                           29,QN
33 | | | | | | (Ey1)¬(P¦(z4)↔P¦(y1))                                                                           30,QN
34 | | | | | | ¬(P¦(y4)↔P¦(z4))                                                                                31,EI
35 | | | | | | ¬(P¦(z4)↔P¦(z4))                                                                                32,EI
36 | | | | | | ¬(P¦(z4)↔P¦(z4))                                                                                33,EI
37 | | | | | | *show ((Ex2)Q¦(x2)↔(Ay2)P¦(y2))
38 | | | | | | | *show ((Ay2)P¦(y2)→(Ex2)Q¦(x2))
39 | | | | | | | | (Ay2)P¦(y2)                                                                                   ASSUME
40 | | | | | | | | *show (Ex2)Q¦(x2)
41 | | | | | | | | | ¬(Ex2)Q¦(x2)                                                                                ASSUME
42 | | | | | | | | | (Ax2)¬Q¦(x2)                                                                                41,QN
43 | | | | | | | | | P¦(y4)                                                                                      39,UI
44 | | | | | | | | | P¦(z4)                                                                                      39,UI
45 | | | | | | | | | *show (P¦(y4)↔P¦(z4))
46 | | | | | | | | | | *show (P¦(z4)→P¦(y4))
47 | | | | | | | | | | | P¦(y4)                                                                                    43,R
```

```
48 | | | | | | | | | | *show (P¦(y₄)→P¦(z₁))                    39,UI
49 | | | | | | | | | | | P¦(z₁)                                  48,46,CB
50 | | | | | | | | | | (P¦(y₄)↔P¦(z₁))                         34,R
51 | | | | | | | | | | ¬(P¦(y₄)↔P¦(z₁))
52 | | | | | | | | | *show ((Ex₂)Q¦(x₂)→(Ay₂)P¦(y₂))           ASSUME
53 | | | | | | | | | | (Ex₂)Q¦(x₂)                             53,EI
54 | | | | | | | | | | *show (Ay₂)P¦(y₂)                       12,UI
55 | | | | | | | | | | | *show P¦(y₂)                          12,UI
56 | | | | | | | | | | | | ¬P¦(y₂)                             27,UI
57 | | | | | | | | | | | | Q¦(z₄)                              27,UI
58 | | | | | | | | | | | | (Q¦(z₄)↔Q¦(y₂))                     58,BC
59 | | | | | | | | | | | | (Q¦(z₄)↔Q¦(z₄))                     58,BC
60 | | | | | | | | | | | | ¬(Ay₂)(P¦(y₂)↔P¦(y₂))                59,BC
61 | | | | | | | | | | | | ¬(Ay₂)(P¦(z₄)↔P¦(y₂))                59,BC
62 | | | | | | | | | | | | (Q¦(z₄)→Q¦(y₂))                     65,57,MP
63 | | | | | | | | | | | | (Q¦(y₂)→Q¦(z₄))                     22,R
64 | | | | | | | | | | | | (Q¦(z₄)→Q¦(z₄))                     52,38,CB
65 | | | | | | | | | | | | (Q¦(z₄)→Q¦(z₄))                     10,24,MP
66 | | | | | | | | | | | | Q¦(z₄)                              24,EI
67 | | | | | | | | | | | | ¬Q¦(z₄)                             12,UI
68 | | | | | | | | | | ((Ex₂)Q¦(x₂)↔(Ay₂)P¦(y₂))                70,UI
69 | | | | | | | | | ((Ex₂)Q¦(x₂)↔(Ay₂)P¦(y₂))                70,UI
70 | | | | | | | | | (Ay₂)(P¦(z₃)↔P¦(y₂))                    70,UI
71 | | | | | | | | | (Q¦(z₄)↔Q¦(z₃))                         70,UI
72 | | | | | | | | | (P¦(z₃)↔P¦(y₄))                         69,BC
73 | | | | | | | | | (P¦(z₃)↔P¦(z₄))                         69,BC
74 | | | | | | | | | (P¦(z₃)↔P¦(z₄))                         71,BC
75 | | | | | | | | | (P¦(z₃)↔P¦(z₃))                         71,BC
76 | | | | | | | | | ((Ex₂)Q¦(x₂)→(Ay₂)P¦(y₂))                72,BC
77 | | | | | | | | | ((Ay₂)P¦(y₂)→(Ex₂)Q¦(x₂))                72,BC
78 | | | | | | | | | (Q¦(z₄)→Q¦(z₃))                         73,BC
79 | | | | | | | | | (Q¦(z₃)→Q¦(z₄))                         73,BC
80 | | | | | | | | | (P¦(z₃)→P¦(y₄))                         74,BC
81 | | | | | | | | | (P¦(y₄)→P¦(z₃))                         74,BC
82 | | | | | | | | | (P¦(z₃)→P¦(z₄))                         75,BC
83 | | | | | | | | | (P¦(z₄)→P¦(z₃))                         85,13,MP
84 | | | | | | | | | (P¦(z₃)→P¦(z₄))                         22,79,MT
85 | | | | | | | | | (P¦(z₄)→P¦(z₃))                         80,87,MP
86 | | | | | | | | | (P¦(z₃)→P¦(z₃))                         82,87,MP
87 | | | | | | | | | P¦(z₃)
88 | | | | | | | | | ¬Q¦(z₃)                                 ASSUME
89 | | | | | | | | | P¦(y₄)
90 | | | | | | | | | P¦(z₄)
91 | | | | | | | | | *show Q¦(z₄)                            ASSUME
92 | | | | | | | | | | ¬Q¦(z₄)
93 | | | | | | | | | | *show Q¦(z₄)
94 | | | | | | | | | | | ¬Q¦(z₄)
```

```
 95 | | | | | | *show (Ex1)Q¦(x1)                                    ASSUME
 96 | | | | | |  ¬(Ex1)Q¦(x1)                                     96,77,MT
 97 | | | | | |  ¬(Ay1)P¦(y1)                                     96,QN
 98 | | | | | |  (Ax1)¬Q¦(x1)                                     97,QN
 99 | | | | | |  (Ey1)¬P¦(y1)                                     99,EI
100 | | | | | |  ¬P¦(z1)                                          12,UI
101 | | | | | |  (Q¦(z1)↔Q¦(z1))                                  70,UI
102 | | | | | |  (P¦(z1)↔P¦(z1))                                  98,UI
103 | | | | | |  ¬Q¦(z1)                                          101,BC
104 | | | | | |  (Q¦(z1)→Q¦(z1))                                  101,BC
105 | | | | | |  (Q¦(z1)→Q¦(z1))                                  102,BC
106 | | | | | |  (P¦(z1)→P¦(z1))                                  102,BC
107 | | | | | |  (P¦(z1)→P¦(z1))                                  106,87,MP
108 | | | | | |  P¦(z1)                                           76,95,MP
109 | | | | | (Ay1)P¦(y1)                                         95,EI
110 | | | | | Q¦(z1)                                              12,UI
111 | | | | | (Q¦(z1)↔Q¦(z1))                                     70,UI
112 | | | | | (P¦(z1)↔P¦(z1))                                     109,UI
113 | | | | | P¦(z1)                                              111,BC
114 | | | | | (Q¦(z1)→Q¦(z1))                                     111,BC
115 | | | | | (Q¦(z1)→Q¦(z1))                                     112,BC
116 | | | | | (P¦(z1)→P¦(z1))                                     112,BC
117 | | | | | (P¦(z1)→P¦(z1))                                     115,110,MP
118 | | | | | Q¦(z1)                                              92,R
119 | | | | |  ¬Q¦(z1)                                          23,R
120 | | | | ¬Q¦(z1)
```

CPU time: 754 msec
Statements: 59456

```
1  *show (((Ex1)(Ay1)(Q1(x1)↔Q1(y1))↔((Ex2)P1(x2)↔(Ay2)Q1(y2)))→((Ex1)(Ay1)(P1(x1)↔P1(y1))→((Ex2)Q1(x2)↔(Ay2)P1(y2))))
2  | ((Ex1)(Ay1)(Q1(x1)↔Q1(y1))↔((Ex2)P1(x2)↔(Ay2)Q1(y2)))                                                    ASSUME
3  | *show ((Ex1)(Ay1)(P1(x1)↔P1(y1))→((Ex2)Q1(x2)↔(Ay2)P1(y2)))
4  | | (Ex1)(Ay1)(P1(x1)↔P1(y1))                                                                             ASSUME
5  | | *show ((Ex1)Q1(x1)↔(Ay1)P1(y1))
6  | | | *show ((Ay1)P1(y1)→(Ex1)Q1(x1))
7  | | | | (Ay1)P1(y1)                                                                                         ASSUME
8  | | | | *show (Ex1)Q1(x1)
9  | | | | | ¬(Ex1)Q1(x1)                                                                                      ASSUME
10 | | | | | ((Ex1)(Ay1)(Q1(x1)↔Q1(y1))→((Ex2)P1(x2)↔(Ay2)Q1(y2)))                                              2,BC
11 | | | | | (((Ex2)P1(x2)↔(Ay2)Q1(y2))→(Ex1)(Ay1)(Q1(x1)↔Q1(y1)))                                              2,BC
12 | | | | | (Ay1)(P1(v2)↔P1(y1))                                                                              4,EI
13 | | | | | (Ax1)¬Q1(x1)                                                                                      9,QN
14 | | | | | P1(v2)                                                                                            7,UI
15 | | | | | (P1(v2)↔P1(v2))                                                                                   12,UI
16 | | | | | ¬Q1(v2)                                                                                           13,UI
17 | | | | | (P1(v2)→P1(v2))                                                                                   15,BC
18 | | | | | *show (Ex1)(Ay1)(Q1(x1)↔Q1(y1))
19 | | | | | | ¬(Ex1)(Ay1)(Q1(x1)↔Q1(y1))                                                                      ASSUME
20 | | | | | | ¬((Ex2)P1(x2)↔(Ay2)Q1(y2))                                                                      19,11,MT
21 | | | | | | (Ax1)¬(Ay1)(Q1(x1)↔Q1(y1))                                                                      19,QN
22 | | | | | | ¬(Ay1)(Q1(v2)↔Q1(y1))                                                                           21,UI
23 | | | | | | (Ey1)¬(Q1(v2)↔Q1(y1))                                                                           22,QN
24 | | | | | | ¬(Q1(v2)↔Q1(v2))                                                                                23,EI
25 | | | | | | P1(v2)                                                                                          7,UI
26 | | | | | | (P1(v2)↔P1(v2))                                                                                 12,UI
27 | | | | | | ¬Q1(v2)                                                                                         13,UI
28 | | | | | | (P1(v2)→P1(v2))                                                                                 26,BC
29 | | | | | | (P1(v2)→P1(v2))                                                                                 26,BC
30 | | | | | | *show ((Ex2)P1(x2)↔(Ay2)Q1(y2))
31 | | | | | | | *show ((Ay2)Q1(y2)→(Ex2)P1(x2))
32 | | | | | | | | (Ex2)P1(x2)                                                                                   25,EG
33 | | | | | | | *show ((Ex2)P1(x2)→(Ay2)Q1(y2))
34 | | | | | | | | (Ex2)P1(x2)                                                                                   ASSUME
35 | | | | | | | | *show (Ay2)Q1(y2)
36 | | | | | | | | | *show Q1(y2)
37 | | | | | | | | | | ¬Q1(y2)                                                                                     ASSUME
38 | | | | | | | | | | P1(v1)                                                                                      34,EI
39 | | | | | | | | | | P1(y2)                                                                                      7,UI
40 | | | | | | | | | | (P1(v2)↔P1(y2))                                                                             12,UI
41 | | | | | | | | | | (P1(v2)↔P1(v1))                                                                             12,UI
42 | | | | | | | | | | ¬Q1(v1)                                                                                     13,UI
43 | | | | | | | | | | ¬(Ay1)(Q1(y2)↔Q1(y1))                                                                       21,UI
44 | | | | | | | | | | ¬(Ay1)(Q1(v1)↔Q1(y1))                                                                       21,UI
45 | | | | | | | | | | (P1(v2)→P1(y2))                                                                             40,BC
46 | | | | | | | | | | (P1(y2)→P1(v2))                                                                             40,BC
47 | | | | | | | | | | (P1(v2)→P1(v1))                                                                             41,BC
```

```
48 | | | | | | | | | | (P¦(v₁)→P¦(v₂))                                41,BC
49 | | | | | | | | | | (Ey₁)¬(Q¦(y₂)↔Q¦(y₁))                          43,QN
50 | | | | | | | | | | (Ey₂)¬(Q¦(v₁)↔Q¦(y₂))                          44,QN
51 | | | | | | | | | | ¬(Q¦(y₂)↔Q¦(v₂))                               49,EI
52 | | | | | | | | | | ¬(Q¦(v₁)↔Q¦(v₂))                               50,EI
53 | | | | | | | | | | P¦(v₂)                                         7,UI
54 | | | | | | | | | | P¦(v₃)                                         7,UI
55 | | | | | | | | | | (P¦(v₂)↔P¦(v₂))                                12,UI
56 | | | | | | | | | | (P¦(v₃)↔P¦(v₃))                                12,UI
57 | | | | | | | | | | ¬Q¦(v₂)                                        13,UI
58 | | | | | | | | | | ¬Q¦(v₃)                                        13,UI
59 | | | | | | | | | | (P¦(v₂)→P¦(v₂))                                55,BC
60 | | | | | | | | | | (P¦(v₂)→P¦(v₃))                                55,BC
61 | | | | | | | | | | (P¦(v₃)→P¦(v₂))                                56,BC
62 | | | | | | | | | | (P¦(v₃)→P¦(v₃))                                56,BC
63 | | | | | | | | | | *show (Q¦(v₂)↔Q¦(v₂))
64 | | | | | | | | | | | *show (Q¦(v₂)→Q¦(v₂))
65 | | | | | | | | | | | | Q¦(v₂)                                       ASSUME
66 | | | | | | | | | | | | ¬Q¦(v₂)                                      27,R
67 | | | | | | | | | | | *show (Q¦(v₂)→Q¦(v₂))
68 | | | | | | | | | | | | Q¦(v₂)                                       ASSUME
69 | | | | | | | | | | | | ¬Q¦(v₂)                                      16,R
70 | | | | | | | | | | | (Q¦(v₂)↔Q¦(v₂))                                67,64,CB
71 | | | | | | | | | | ¬(Q¦(v₂)↔Q¦(v₂))                               24,R
72 | | | | | | | | ((Ex₂)P¦(x₂)↔(Ay₂)Q¦(y₂))                       33,31,CB
73 | | | | | | | ((Ex₂)P¦(x₂)↔(Ay₂)Q¦(y₂))                       10,18,MP
74 | | | | | | | (Ay₁)(Q¦(v₂)↔Q¦(y₁))                            18,EI
75 | | | | | | | P¦(v₂)                                         7,UI
76 | | | | | | | (P¦(v₂)↔P¦(v₂))                                12,UI
77 | | | | | | | ¬Q¦(v₂)                                        13,UI
78 | | | | | | | (Q¦(v₂)↔Q¦(v₂))                                74,UI
79 | | | | | | | (Q¦(v₃)↔Q¦(v₂))                                74,UI
80 | | | | | | | ((Ex₂)P¦(x₂)→(Ay₂)Q¦(y₂))                       73,BC
81 | | | | | | | ((Ay₂)Q¦(y₂)→(Ex₂)P¦(x₂))                       73,BC
82 | | | | | | | (P¦(v₂)→P¦(v₂))                                76,BC
83 | | | | | | | (P¦(v₂)→P¦(v₃))                                76,BC
84 | | | | | | | (Q¦(v₂)→Q¦(v₂))                                78,BC
85 | | | | | | | (Q¦(v₂)→Q¦(v₃))                                78,BC
86 | | | | | | | (Q¦(v₃)→Q¦(v₂))                                79,BC
87 | | | | | | | *show (Ex₂)P¦(x₂)
88 | | | | | | | | (Ex₂)P¦(x₂)                                  75,EG
89 | | | | | | | (Ay₂)Q¦(y₂)                                    80,87,MP
90 | | | | | | | P¦(v₃)                                         87,EI
91 | | | | | | | (P¦(v₂)↔P¦(v₃))                                12,UI
92 | | | | | | | ¬Q¦(v₃)                                        13,UI
93 | | | | | | | (Q¦(v₂)↔Q¦(v₃))                                74,UI
94 | | | | | | | Q¦(v₂)                                         89,UI
```

```
95  | | | *show ((Ex₂)Q¦(x₂)→(Ay₂)P¦(y₂))
96  | | |  (Ex₂)Q¦(x₂)                                                                                                    ASSUME
97  | | |  *show (Ay₂)P¦(y₂)
98  | | |   *show P¦(y₂)
99  | | |    ¬P¦(y₂)                                                                                                      ASSUME
100 | | |    ((Ex₂)(Ay₂)(Q¦(x₂)↔Q¦(y₂))→((Ex₄)P¦(x₄)↔(Ay₄)Q¦(y₄)))                                                          2,BC
101 | | |    (((Ex₄)P¦(x₄)↔(Ay₄)Q¦(y₄))→(Ex₂)(Ay₂)(Q¦(x₂)↔Q¦(y₂)))                                                          2,BC
102 | | |    (Ay₂)(P¦(v₂)↔P¦(y₁))                                                                                         4,EI
103 | | |    Q¦(v₁)                                                                                                       96,EI
104 | | |    (P¦(v₂)↔P¦(y₂))                                                                                              102,UI
105 | | |    (P¦(v₂)↔P¦(v₂))                                                                                              102,UI
106 | | |    (P¦(v₂)↔P¦(v₁))                                                                                              102,UI
107 | | |    (P¦(v₂)→P¦(y₂))                                                                                              104,BC
108 | | |    (P¦(y₂)→P¦(v₂))                                                                                              104,BC
109 | | |    (P¦(v₂)→P¦(v₁))                                                                                              105,BC
110 | | |    (P¦(v₂)→P¦(v₁))                                                                                              106,BC
111 | | |    (P¦(v₁)→P¦(v₂))                                                                                              106,BC
112 | | |    ¬P¦(v₂)                                                                                                      99,107,MT
113 | | |    ¬P¦(v₁)                                                                                                      112,111,MT
114 | | |    *show (Ex₂)(Ay₂)(Q¦(x₂)↔Q¦(y₂))
115 | | |     ¬(Ex₂)(Ay₂)(Q¦(x₂)↔Q¦(y₂))                                                                                  ASSUME
116 | | |     ¬((Ex₄)P¦(x₄)↔(Ay₄)Q¦(y₄))                                                                                  115,101,MT
117 | | |     (Ax₂)¬(Ay₂)(Q¦(x₂)↔Q¦(y₂))                                                                                  115,QN
118 | | |     ¬(Ay₂)(Q¦(y₂)↔Q¦(y₂))                                                                                       117,UI
119 | | |     ¬(Ay₂)(Q¦(v₂)↔Q¦(y₂))                                                                                       117,UI
120 | | |     ¬(Ay₂)(Q¦(v₁)↔Q¦(y₂))                                                                                       117,UI
121 | | |     (Ey₂)¬(Q¦(y₂)↔Q¦(y₂))                                                                                       118,QN
122 | | |     (Ey₂)¬(Q¦(v₂)↔Q¦(y₂))                                                                                       119,QN
123 | | |     (Ey₂)¬(Q¦(v₁)↔Q¦(y₂))                                                                                       120,QN
124 | | |     ¬(Q¦(y₂)↔Q¦(v₀))                                                                                            121,EI
125 | | |     ¬(Q¦(v₂)↔Q¦(u₀))                                                                                            122,EI
126 | | |     ¬(Q¦(v₁)↔Q¦(u₀))                                                                                            123,EI
127 | | |     (P¦(v₂)↔P¦(v₀))                                                                                             102,UI
128 | | |     (P¦(v₂)↔P¦(u₀))                                                                                             102,UI
129 | | |     (P¦(v₂)↔P¦(u₀))                                                                                             102,UI
130 | | |     (P¦(v₂)→P¦(v₀))                                                                                             127,BC
131 | | |     (P¦(v₀)→P¦(v₂))                                                                                             127,BC
132 | | |     (P¦(v₂)→P¦(u₀))                                                                                             128,BC
133 | | |     (P¦(u₀)→P¦(v₂))                                                                                             128,BC
134 | | |     (P¦(v₂)→P¦(u₀))                                                                                             129,BC
135 | | |     (P¦(u₀)→P¦(v₁))                                                                                             129,BC
136 | | |     ¬P¦(v₀)                                                                                                     112,131,MT
137 | | |     ¬P¦(u₀)                                                                                                     112,133,MT
138 | | |     ¬P¦(u₀)                                                                                                     112,135,MT
139 | | |     *show ((Ex₄)P¦(x₄)↔(Ay₄)Q¦(y₄))
140 | | |      *show ((Ay₄)Q¦(y₄)→(Ex₄)P¦(x₄))
141 | | |       (Ay₄)Q¦(y₄)                                                                                             ASSUME
```

```
142 | | | | | | | | | *show (Ex.)P¦(x.)                                  
143 | | | | | | | | | | ¬(Ex.)P¦(x.)                                    ASSUME
144 | | | | | | | | | | (Ax.)¬P¦(x.)                                    143,QN
145 | | | | | | | | | | Q¦(y.)                                          141,UI
146 | | | | | | | | | | Q¦(v.)                                          141,UI
147 | | | | | | | | | | Q¦(v.)                                          141,UI
148 | | | | | | | | | | Q¦(u.)                                          141,UI
149 | | | | | | | | | | Q¦(u.)                                          141,UI
150 | | | | | | | | | | *show (Q¦(y.)↔Q¦(v.))
151 | | | | | | | | | | | *show (Q¦(v.)→Q¦(y.))
152 | | | | | | | | | | | | Q¦(y.)                                        145,R
153 | | | | | | | | | | | *show (Q¦(y.)→Q¦(v.))
154 | | | | | | | | | | | | Q¦(v.)                                        147,R
155 | | | | | | | | | | | (Q¦(y.)↔Q¦(v.))                                 153,151,CB
156 | | | | | | | | | | ¬(Q¦(y.)↔Q¦(v.))                                  124,R
157 | | | | | | | | | *show ((Ex.)P¦(x.)→(Ay.)Q¦(y.))
158 | | | | | | | | | | (Ex.)P¦(x.)                                     ASSUME
159 | | | | | | | | | | *show (Ay.)Q¦(y.)
160 | | | | | | | | | | | *show Q¦(y.)
161 | | | | | | | | | | | | ¬Q¦(y.)                                       ASSUME
162 | | | | | | | | | | | | P¦(u.)                                        158,EI
163 | | | | | | | | | | | | (P¦(v.)↔P¦(y.))                               102,UI
164 | | | | | | | | | | | | (P¦(v.)↔P¦(u.))                               102,UI
165 | | | | | | | | | | | | ¬(Ay.)(Q¦(y.)↔Q¦(y.))                          117,UI
166 | | | | | | | | | | | | ¬(Ay.)(Q¦(u.)↔Q¦(y.))                          117,UI
167 | | | | | | | | | | | | (P¦(v.)→P¦(y.))                               163,BC
168 | | | | | | | | | | | | (P¦(y.)→P¦(v.))                               163,BC
169 | | | | | | | | | | | | (P¦(v.)→P¦(u.))                               164,BC
170 | | | | | | | | | | | | (P¦(u.)→P¦(v.))                               164,BC
171 | | | | | | | | | | | | P¦(v.)                                        170,162,MP
172 | | | | | | | | | | | | ¬P¦(v.)                                       112,R
173 | | | | | | | | | | ((Ex.)P¦(x.)↔(Ay.)Q¦(y.))                          157,140,CB
174 | | | | | | | | ((Ex.)P¦(x.)↔(Ay.)Q¦(y.))                              100,114,MP
175 | | | | | | | | (Ay.)(Q¦(u.)↔Q¦(y.))                                   114,EI
176 | | | | | | | | (P¦(v.)↔P¦(u.))                                       102,UI
177 | | | | | | | | (Q¦(u.)↔Q¦(y.))                                       175,UI
178 | | | | | | | | (Q¦(u.)↔Q¦(v.))                                       175,UI
179 | | | | | | | | (Q¦(u.)↔Q¦(v.))                                       175,UI
180 | | | | | | | | (Q¦(u.)↔Q¦(u.))                                       175,UI
181 | | | | | | | | ((Ex.)P¦(x.)→(Ay.)Q¦(y.))                              174,BC
182 | | | | | | | | ((Ay.)Q¦(y.)→(Ex.)P¦(x.))                              174,BC
183 | | | | | | | | (P¦(v.)→P¦(u.))                                       176,BC
184 | | | | | | | | (P¦(u.)→P¦(v.))                                       176,BC
185 | | | | | | | | (Q¦(u.)→Q¦(y.))                                       177,BC
186 | | | | | | | | (Q¦(y.)→Q¦(u.))                                       177,BC
187 | | | | | | | | (Q¦(u.)→Q¦(v.))                                       178,BC
188 | | | | | | | | (Q¦(v.)→Q¦(u.))                                       178,BC
```

```
189 | | | | | | (Q¦(u₁)→Q¦(v₁))                                    179,BC
190 | | | | | | (Q¦(v₁)→Q¦(u₁))                                    179,BC
191 | | | | | | (Q¦(u₁)→Q¦(u₁))                                    180,BC
192 | | | | | | Q¦(u₁)                                             190,103,MP
193 | | | | | | ¬P¦(u₁)                                            112,184,MT
194 | | | | | | Q¦(y₂)                                             185,192,MP
195 | | | | | | Q¦(v₂)                                             187,192,MP
196 | | | | | | *show P¦(v₂)
197 | | | | | | | ¬P¦(v₂)                                          ASSUME
198 | | | | | | | *show P¦(v₂)
199 | | | | | | | | ¬P¦(v₂)                                        ASSUME
200 | | | | | | | | *show (Ex₁)P¦(x₁)
201 | | | | | | | | | ¬(Ex₁)P¦(x₁)                                 ASSUME
202 | | | | | | | | | ¬(Ay₁)Q¦(y₁)                                 201,182,MT
203 | | | | | | | | | (Ax₁)¬P¦(x₁)                                 201,QN
204 | | | | | | | | | (Ey₁)¬Q¦(y₁)                                 202,QN
205 | | | | | | | | | ¬Q¦(u₁)                                      204,EI
206 | | | | | | | | | (P¦(v₂)↔P¦(u₁))                              102,UI
207 | | | | | | | | | (Q¦(u₁)↔Q¦(u₁))                              175,UI
208 | | | | | | | | | ¬P¦(u₁)                                      203,UI
209 | | | | | | | | | (P¦(v₂)→P¦(u₁))                              206,BC
210 | | | | | | | | | (P¦(u₁)→P¦(v₂))                              206,BC
211 | | | | | | | | | (Q¦(u₁)→Q¦(u₁))                              207,BC
212 | | | | | | | | | (Q¦(u₁)→Q¦(u₁))                              207,BC
213 | | | | | | | | | Q¦(u₁)                                       211,192,MP
214 | | | | | | | | (Ay₁)Q¦(y₁)                                    181,200,MP
215 | | | | | | | | P¦(u₁)                                         200,EI
216 | | | | | | | | (P¦(v₂)↔P¦(u₁))                                102,UI
217 | | | | | | | | (Q¦(u₁)↔Q¦(u₁))                                175,UI
218 | | | | | | | | Q¦(u₁)                                         214,UI
219 | | | | | | | | (P¦(v₂)→P¦(u₁))                                216,BC
220 | | | | | | | | (P¦(u₁)→P¦(v₂))                                216,BC
221 | | | | | | | | (Q¦(u₁)→Q¦(u₁))                                217,BC
222 | | | | | | | | (Q¦(u₁)→Q¦(u₁))                                217,BC
223 | | | | | | | | P¦(v₂)                                         220,215,MP
224 | | | | | | | | ¬P¦(v₂)                                        197,R
225 | | | | | | | ¬P¦(v₂)                                          113,R
226 | | | | ((Ex₁)Q¦(x₁)↔(Ay₁)P¦(y₁))                                95,6,CB
```

CPU time: 1326 msec
Statements: 100344

```
1  *show (((Ex1)(Ay1)(P¦(x1)↔P¦(y1))↔((Ex2)Q¦(x2)↔(Ay2)P¦(y2)))→((Ex3)(Ay3)(Q¦(x3)↔Q¦(y3))→((Ay4)Q¦(y4)→(Ex4)P¦(x4))))
2  | ((Ex1)(Ay1)(P¦(x1)↔P¦(y1))↔((Ex2)Q¦(x2)↔(Ay2)P¦(y2)))                                                    ASSUME
3  | *show ((Ex3)(Ay3)(Q¦(x3)↔Q¦(y3))→((Ay4)Q¦(y4)→(Ex4)P¦(x4)))
4  | | (Ex3)(Ay3)(Q¦(x3)↔Q¦(y3))                                                                             ASSUME
5  | | *show ((Ay4)Q¦(y4)→(Ex4)P¦(x4))
6  | | | (Ay4)Q¦(y4)                                                                                         ASSUME
7  | | | *show (Ex4)P¦(x4)
8  | | | | ¬(Ex4)P¦(x4)                                                                                      ASSUME
9  | | | | ((Ex1)(Ay1)(P¦(x1)↔P¦(y1))→((Ex2)Q¦(x2)↔(Ay2)P¦(y2)))                                              2,BC
10 | | | | (((Ex3)Q¦(x3)↔(Ay3)P¦(y3))→(Ex1)(Ay1)(P¦(x1)↔P¦(y1)))                                              2,BC
11 | | | | (Ay3)(Q¦(v1)↔Q¦(y3))                                                                               4,EI
12 | | | | (Ax4)¬P¦(x4)                                                                                       8,QN
13 | | | | Q¦(v1)                                                                                             6,UI
14 | | | | (Q¦(v1)↔Q¦(v1))                                                                                    11,UI
15 | | | | ¬P¦(v1)                                                                                            12,UI
16 | | | | (Q¦(v1)→Q¦(v1))                                                                                    14,BC
17 | | | | *show (Ex1)(Ay1)(P¦(x1)↔P¦(y1))
18 | | | | | ¬(Ex1)(Ay1)(P¦(x1)↔P¦(y1))                                                                       ASSUME
19 | | | | | ¬((Ex1)Q¦(x1)↔(Ay1)P¦(y1))                                                                       18,10,MT
20 | | | | | (Ax1)¬(Ay1)(P¦(x1)↔P¦(y1))                                                                       18,QN
21 | | | | | ¬(Ay1)(P¦(v1)↔P¦(y1))                                                                            20,UI
22 | | | | | (Ey1)¬(P¦(v1)↔P¦(y1))                                                                            21,QN
23 | | | | | ¬(P¦(v1)↔P¦(v1))                                                                                 22,EI
24 | | | | | Q¦(v1)                                                                                           6,UI
25 | | | | | (Q¦(v1)↔Q¦(v1))                                                                                  11,UI
26 | | | | | ¬P¦(v1)                                                                                          12,UI
27 | | | | | (Q¦(v1)→Q¦(v1))                                                                                  25,BC
28 | | | | | (Q¦(v1)→Q¦(v1))                                                                                  25,BC
29 | | | | | *show ((Ex2)Q¦(x2)↔(Ay2)P¦(y2))
30 | | | | | | *show ((Ay2)P¦(y2)→(Ex2)Q¦(x2))
31 | | | | | | | (Ex2)Q¦(x2)                                                                                    24,EG
32 | | | | | | *show ((Ex2)Q¦(x2)→(Ay2)P¦(y2))
33 | | | | | | | (Ex2)Q¦(x2)                                                                                    ASSUME
34 | | | | | | | *show (Ay2)P¦(y2)
35 | | | | | | | | *show P¦(y2)
36 | | | | | | | | | ¬P¦(y2)                                                                                      ASSUME
37 | | | | | | | | | Q¦(v2)                                                                                       33,EI
38 | | | | | | | | | Q¦(y2)                                                                                       6,UI
39 | | | | | | | | | (Q¦(v2)↔Q¦(y2))                                                                              11,UI
40 | | | | | | | | | (Q¦(v2)↔Q¦(v2))                                                                              11,UI
41 | | | | | | | | | ¬P¦(v2)                                                                                      12,UI
42 | | | | | | | | | ¬(Ay2)(P¦(y2)↔P¦(y2))                                                                        20,UI
43 | | | | | | | | | ¬(Ay2)(P¦(v2)↔P¦(y2))                                                                        20,UI
44 | | | | | | | | | (Q¦(v2)→Q¦(y2))                                                                              39,BC
45 | | | | | | | | | (Q¦(y2)→Q¦(v2))                                                                              39,BC
46 | | | | | | | | | (Q¦(v2)→Q¦(v2))                                                                              40,BC
47 | | | | | | | | | (Q¦(v2)→Q¦(v2))                                                                              40,BC
```

```
48 | | | | | | | | (Ey1)¬(P¦(y1)↔P¦(y1))
49 | | | | | | | | (Ey1)¬(P¦(v1)↔P¦(y1))
50 | | | | | | | | ¬(P¦(y1)↔P¦(v1))
51 | | | | | | | | ¬(P¦(v1)↔P¦(v1))
52 | | | | | | | | Q¦(v1)
53 | | | | | | | | Q¦(v1)
54 | | | | | | | | (Q¦(v1)↔Q¦(v1))
55 | | | | | | | | (Q¦(v1)↔Q¦(v1))
56 | | | | | | | | ¬P¦(v1)
57 | | | | | | | | ¬P¦(v1)
58 | | | | | | | | (Q¦(v1)→Q¦(v1))
59 | | | | | | | | (Q¦(v1)→Q¦(v1))
60 | | | | | | | | (Q¦(v1)→Q¦(v1))
61 | | | | | | | | (Q¦(v1)→Q¦(v1))
62 | | | | | | | | *show (P¦(v1)↔P¦(v1))
63 | | | | | | | | | *show (P¦(v1)→P¦(v1))
64 | | | | | | | | | | P¦(v1)
65 | | | | | | | | | | ¬P¦(v1)
66 | | | | | | | | | *show (P¦(v1)→P¦(v1))
67 | | | | | | | | | | P¦(v1)
68 | | | | | | | | | | ¬P¦(v1)
69 | | | | | | | | | (P¦(v1)↔P¦(v1))
70 | | | | | | | | ¬(P¦(v1)↔P¦(v1))
71 | | | | | | | ((Ex1)Q¦(x1)↔(Ay1)P¦(y1))
72 | | | | | | ((Ex1)Q¦(x1)↔(Ay1)P¦(y1))
73 | | | | | | (Ay1)(P¦(v1)↔P¦(y1))
74 | | | | | | Q¦(v1)
75 | | | | | | (Q¦(v1)↔Q¦(v1))
76 | | | | | | ¬P¦(v1)
77 | | | | | | (P¦(v1)↔P¦(v1))
78 | | | | | | (P¦(v1)↔P¦(v1))
79 | | | | | | ((Ex1)Q¦(x1)→(Ay1)P¦(y1))
80 | | | | | | ((Ay1)P¦(y1)→(Ex1)Q¦(x1))
81 | | | | | | (Q¦(v1)→Q¦(v1))
82 | | | | | | (Q¦(v1)→Q¦(v1))
83 | | | | | | (P¦(v1)→P¦(v1))
84 | | | | | | (P¦(v1)→P¦(v1))
85 | | | | | | (P¦(v1)→P¦(v1))
86 | | | | | | *show (Ex1)Q¦(x1)
87 | | | | | | | (Ex1)Q¦(x1)
88 | | | | | | (Ay1)P¦(y1)
89 | | | | | | Q¦(v1)
90 | | | | | | (Q¦(v1)↔Q¦(v1))
91 | | | | | | ¬P¦(v1)
92 | | | | | | (P¦(v1)↔P¦(v1))
93 | | | | | | P¦(v1)
```

```
42,QN
43,QN
48,EI
49,EI
6,UI
6,UI
11,UI
11,UI
12,UI
12,UI
54,BC
54,BC
55,BC
55,BC

ASSUME
26,R

ASSUME
15,R
66,63,CB
23,R
32,30,CB
9,17,MP
17,EI
6,UI
11,UI
12,UI
73,UI
73,UI
72,BC
72,BC
75,BC
75,BC
77,BC
77,BC
78,BC

74,EG
79,86,MP
86,EI
11,UI
12,UI
73,UI
88,UI
```

CPU time: 543 msec
Statements: 40597

```
1  *show (((Ex1)(Ay1)(Q¦(x1)↔Q¦(y1))↔((Ex2)P¦(x2)↔(Ay2)Q¦(y2)))↔(((Ex1)Q¦(x1)↔(Ay1)P¦(y1))↔(Ex1)(Ay1)(P¦(x1)→P¦(y1))))
2  | ((Ex1)(Ay1)(Q¦(x1)↔Q¦(y1))↔((Ex2)P¦(x2)↔(Ay2)Q¦(y2)))                                                    ASSUME
3  | *show (((Ex1)Q¦(x1)↔(Ay1)P¦(y1))→(Ex1)(Ay1)(P¦(x1)→P¦(y1)))
4  | | ((Ex1)Q¦(x1)↔(Ay1)P¦(y1))                                                                             ASSUME
5  | | *show (Ex1)(Ay1)(P¦(x1)→P¦(y1))
6  | | | ¬(Ex1)(Ay1)(P¦(x1)→P¦(y1))                                                                          ASSUME
7  | | | ((Ex1)(Ay1)(Q¦(x1)↔Q¦(y1))→((Ex2)P¦(x2)↔(Ay2)Q¦(y2)))                                               2,BC
8  | | | (((Ex2)P¦(x2)↔(Ay2)Q¦(y2))→(Ex1)(Ay1)(Q¦(x1)↔Q¦(y1)))                                               2,BC
9  | | | ((Ex1)Q¦(x1)→(Ay1)P¦(y1))                                                                           4,BC
10 | | | ((Ay1)P¦(y1)→(Ex1)Q¦(x1))                                                                           4,BC
11 | | | (Ax1)¬(Ay1)(P¦(x1)→P¦(y1))                                                                          6,QN
12 | | | ¬(Ay1)(P¦(z1)→P¦(y1))                                                                               11,UI
13 | | | (Ey1)¬(P¦(z1)→P¦(y1))                                                                               12,QN
14 | | | ¬(P¦(z1)→P¦(z1))                                                                                    13,EI
15 | | | *show (P¦(z1)→P¦(z1))
16 | | | | P¦(z1)                                                                                            ASSUME
17 | | | | *show P¦(z1)
18 | | | | | ¬P¦(z1)                                                                                         ASSUME
19 | | | | | *show (Ex1)(Ay1)(Q¦(x1)↔Q¦(y1))
20 | | | | | | ¬(Ex1)(Ay1)(Q¦(x1)↔Q¦(y1))                                                                     ASSUME
21 | | | | | | ¬((Ex2)P¦(x2)↔(Ay2)Q¦(y2))                                                                     20,B,MT
22 | | | | | | (Ax1)¬(Ay1)(Q¦(x1)↔Q¦(y1))                                                                     20,QN
23 | | | | | | ¬(Ay1)(Q¦(z1)↔Q¦(y1))                                                                          22,UI
24 | | | | | | (Ey1)¬(Q¦(z1)↔Q¦(y1))                                                                          23,QN
25 | | | | | | ¬(Q¦(z1)↔Q¦(z1))                                                                               24,EI
26 | | | | | | *show ((Ex2)P¦(x2)↔(Ay2)Q¦(y2))
27 | | | | | | | *show ((Ay2)Q¦(y2)→(Ex2)P¦(x2))
28 | | | | | | | | (Ex2)P¦(x2)                                                                                   16,EG
29 | | | | | | | *show ((Ex2)P¦(x2)→(Ay2)Q¦(y2))
30 | | | | | | | | (Ex2)P¦(x2)                                                                                   ASSUME
31 | | | | | | | | *show (Ay2)Q¦(y2)
32 | | | | | | | | | *show Q¦(y2)
33 | | | | | | | | | | ¬Q¦(y2)                                                                                   ASSUME
34 | | | | | | | | | | P¦(z1)                                                                                    30,EI
35 | | | | | | | | | | ¬(Ay1)(P¦(y2)→P¦(y1))                                                                     11,UI
36 | | | | | | | | | | ¬(Ay1)(P¦(z1)→P¦(y1))                                                                     11,UI
37 | | | | | | | | | | ¬(Ay1)(Q¦(y2)↔Q¦(y1))                                                                     22,UI
38 | | | | | | | | | | ¬(Ay1)(Q¦(z1)↔Q¦(y1))                                                                     22,UI
39 | | | | | | | | | | (Ey1)¬(P¦(y2)↔P¦(y1))                                                                     35,QN
40 | | | | | | | | | | (Ey1)¬(P¦(z1)↔P¦(y1))                                                                     36,QN
41 | | | | | | | | | | (Ey1)¬(Q¦(y2)↔Q¦(y1))                                                                     37,QN
42 | | | | | | | | | | (Ey1)¬(Q¦(z1)↔Q¦(y1))                                                                     38,QN
43 | | | | | | | | | | ¬(P¦(y2)→P¦(z1))                                                                          39,EI
44 | | | | | | | | | | ¬(P¦(z1)→P¦(z1))                                                                          40,EI
45 | | | | | | | | | | ¬(Q¦(y2)↔Q¦(z1))                                                                          41,EI
46 | | | | | | | | | | ¬(Q¦(z1)↔Q¦(z1))                                                                          42,EI
47 | | | | | | | | | | *show (P¦(y2)→P¦(z1))
```

```
48 | | | | | | | | | | P¦(y₀)                                                    ASSUME
49 | | | | | | | | | | *show P¦(z₀)
50 | | | | | | | | | |  ¬P¦(z₀)                                                  ASSUME
51 | | | | | | | | | |  *show (P¦(z₀)→P¦(z₀))
52 | | | | | | | | | |  | P¦(z₀)                                                 ASSUME
53 | | | | | | | | | |  | *show P¦(z₀)
54 | | | | | | | | | |  | ¬P¦(z₀)                                                ASSUME
55 | | | | | | | | | |  | *show (Q¦(z₀)↔Q¦(z₁))
56 | | | | | | | | | |  | | *show (Q¦(z₁)→Q¦(z₀))
57 | | | | | | | | | |  | | Q¦(z₁)                                               ASSUME
58 | | | | | | | | | |  | | *show Q¦(z₀)
59 | | | | | | | | | |  | | ¬Q¦(z₀)                                              ASSUME
60 | | | | | | | | | |  | | *show (Q¦(y₀)↔Q¦(z₁))
61 | | | | | | | | | |  | | | *show (Q¦(z₁)→Q¦(y₀))
62 | | | | | | | | | |  | | | Q¦(z₁)                                             ASSUME
63 | | | | | | | | | |  | | | *show (Q¦(z₀)↔Q¦(z₁))
64 | | | | | | | | | |  | | | | *show (Q¦(z₁)→Q¦(z₀))
65 | | | | | | | | | |  | | | | Q¦(z₁)                                           ASSUME
66 | | | | | | | | | |  | | | | *show Q¦(z₀)
67 | | | | | | | | | |  | | | | ¬Q¦(z₀)                                          ASSUME
68 | | | | | | | | | |  | | | | *show (Ex₁)Q¦(x₁)
69 | | | | | | | | | |  | | | | | (Ex₁)Q¦(x₁)                                    65,EG
70 | | | | | | | | | |  | | | | (Ay₁)P¦(y₁)                                      9,68,MP
71 | | | | | | | | | |  | | | | Q¦(z₁)                                           68,EI
72 | | | | | | | | | |  | | | | ¬(Ay₁)(P¦(z₁)→P¦(y₁))                            11,UI
73 | | | | | | | | | |  | | | | ¬(Ay₁)(Q¦(z₁)↔Q¦(y₁))                            22,UI
74 | | | | | | | | | |  | | | | P¦(z₁)                                           70,UI
75 | | | | | | | | | |  | | | | (Ey₁)¬(P¦(z₁)→P¦(y₁))                            72,QN
76 | | | | | | | | | |  | | | | (Ey₁)¬(Q¦(z₁)↔Q¦(y₁))                            73,QN
77 | | | | | | | | | |  | | | | ¬(P¦(z₁)→P¦(z₀))                                 75,EI
78 | | | | | | | | | |  | | | | ¬(Q¦(z₁)↔Q¦(y₀))                                 76,EI
79 | | | | | | | | | |  | | | | *show (P¦(z₁)→P¦(z₀))
80 | | | | | | | | | |  | | | | | P¦(z₀)                                         70,UI
81 | | | | | | | | | |  | | | *show (Q¦(z₀)→Q¦(z₁))
82 | | | | | | | | | |  | | | Q¦(z₀)                                             ASSUME
83 | | | | | | | | | |  | | | *show Q¦(z₁)
84 | | | | | | | | | |  | | | ¬Q¦(z₁)                                            ASSUME
85 | | | | | | | | | |  | | | *show (Ex₁)Q¦(x₁)
86 | | | | | | | | | |  | | | | (Ex₁)Q¦(x₁)                                    82,EG
87 | | | | | | | | | |  | | | (Ay₁)P¦(y₁)                                      9,85,MP
88 | | | | | | | | | |  | | | Q¦(y₀)                                           85,EI
89 | | | | | | | | | |  | | | ¬(Ay₁)(P¦(y₀)→P¦(y₁))                            11,UI
90 | | | | | | | | | |  | | | ¬(Ay₁)(Q¦(y₀)↔Q¦(y₁))                            22,UI
91 | | | | | | | | | |  | | | P¦(y₀)                                           87,UI
92 | | | | | | | | | |  | | | (Ey₁)¬(P¦(y₀)→P¦(y₁))                            89,QN
93 | | | | | | | | | |  | | | (Ey₁)¬(Q¦(y₀)↔Q¦(y₁))                            90,QN
94 | | | | | | | | | |  | | | ¬(P¦(y₀)→P¦(y₁))                                 92,EI
```

```
95  | | |
```

```
142 | | |
```

```
189 | | | | | | | | *show (P¦(w₀)→P¦(w₀))
190 | | | | | | | |  P¦(w₀)                          ASSUME
191 | | | | | | | |  ¬P¦(w₀)                         181,R
192 | | | | | | (Ay₂)P¦(y₂)                        9,174,MP
193 | | | | | | P¦(x₀)                             192,UI
```

CPU time: 1010 msec
Statements: 79521

```
1  *show (((Ex1)(Ay1)(P¦(x1)↔P¦(y1))↔((Ex2)Q¦(x2)↔(Ay2)P¦(y2)))→(((Ex4)P¦(x4)↔(Ay4)Q¦(y4))→(Ex3)(Ay3)(Q¦(x3)→Q¦(y3))))
2  | ((Ex1)(Ay1)(P¦(x1)↔P¦(y1))↔((Ex2)Q¦(x2)↔(Ay2)P¦(y2)))                                                    ASSUME
3  | *show (((Ex4)P¦(x4)↔(Ay4)Q¦(y4))→(Ex3)(Ay3)(Q¦(x3)→Q¦(y3)))
4  | | ((Ex4)P¦(x4)↔(Ay4)Q¦(y4))                                                                             ASSUME
5  | | *show (Ex3)(Ay3)(Q¦(x3)→Q¦(y3))
6  | | | ¬(Ex3)(Ay3)(Q¦(x3)→Q¦(y3))                                                                          ASSUME
7  | | | ((Ex1)(Ay1)(P¦(x1)↔P¦(y1)))→((Ex2)Q¦(x2)↔(Ay2)P¦(y2)))                                              2,BC
8  | | | (((Ex2)Q¦(x2)↔(Ay2)P¦(y2)))→(Ex1)(Ay1)(P¦(x1)↔P¦(y1)))                                              2,BC
9  | | | ((Ex4)P¦(x4)→(Ay4)Q¦(y4))                                                                           4,BC
10 | | | ((Ay4)Q¦(y4)→(Ex4)P¦(x4))                                                                           4,BC
11 | | | (Ax3)¬(Ay3)(Q¦(x3)→Q¦(y3))                                                                          6,QN
12 | | | ¬(Ay3)(Q¦(z3)→Q¦(y3))                                                                               11,UI
13 | | | (Ey3)¬(Q¦(z3)→Q¦(y3))                                                                               12,QN
14 | | | ¬(Q¦(z3)→Q¦(z4))                                                                                    13,EI
15 | | | *show (Q¦(z3)→Q¦(z4))
16 | | | | Q¦(z4)                                                                                            ASSUME
17 | | | | *show Q¦(z4)
18 | | | | | ¬Q¦(z4)                                                                                         ASSUME
19 | | | | | *show (Ex1)(Ay1)(P¦(x1)↔P¦(y1))
20 | | | | | | ¬(Ex1)(Ay1)(P¦(x1)↔P¦(y1))                                                                    ASSUME
21 | | | | | | ¬((Ex2)Q¦(x2)↔(Ay2)P¦(y2))                                                                    20,8,MT
22 | | | | | | (Ax1)¬(Ay1)(P¦(x1)↔P¦(y1))                                                                    20,QN
23 | | | | | | ¬(Ay1)(P¦(z4)↔P¦(y1))                                                                         22,UI
24 | | | | | | (Ey1)¬(P¦(z4)↔P¦(y1))                                                                         23,QN
25 | | | | | | ¬(P¦(z4)↔P¦(z4))                                                                              24,EI
26 | | | | | | *show ((Ex2)Q¦(x2)↔(Ay2)P¦(y2))
27 | | | | | | | *show ((Ay2)P¦(y2)→(Ex2)Q¦(x2))
28 | | | | | | | | (Ex2)Q¦(x2)                                                                                 16,EG
29 | | | | | | | *show ((Ex2)Q¦(x2)→(Ay2)P¦(y2))
30 | | | | | | | | (Ex2)Q¦(x2)                                                                                 ASSUME
31 | | | | | | | | *show (Ay1)P¦(y2)
32 | | | | | | | | | *show P¦(y2)
33 | | | | | | | | | | ¬P¦(y2)                                                                                 ASSUME
34 | | | | | | | | | | Q¦(z4)                                                                                  30,EI
35 | | | | | | | | | | ¬(Ay1)(Q¦(y2)→Q¦(y1))                                                                   11,UI
36 | | | | | | | | | | ¬(Ay1)(Q¦(z4)→Q¦(y1))                                                                   11,UI
37 | | | | | | | | | | ¬(Ay1)(P¦(y2)↔P¦(y1))                                                                   22,UI
38 | | | | | | | | | | ¬(Ay1)(P¦(z4)↔P¦(y1))                                                                   22,UI
39 | | | | | | | | | | (Ey1)¬(Q¦(y2)→Q¦(y1))                                                                   35,QN
40 | | | | | | | | | | (Ey1)¬(Q¦(z4)→Q¦(y1))                                                                   36,QN
41 | | | | | | | | | | (Ey1)¬(P¦(y2)↔P¦(y1))                                                                   37,QN
42 | | | | | | | | | | (Ey1)¬(P¦(z4)↔P¦(y1))                                                                   38,QN
43 | | | | | | | | | | ¬(Q¦(y2)→Q¦(z4))                                                                        39,EI
44 | | | | | | | | | | ¬(Q¦(z4)→Q¦(z4))                                                                        40,EI
45 | | | | | | | | | | ¬(P¦(y2)↔P¦(z2))                                                                        41,EI
46 | | | | | | | | | | ¬(P¦(z4)↔P¦(z2))                                                                        42,EI
47 | | | | | | | | | | *show (Q¦(y2)→Q¦(z4))
```

```
48  | | | | | | | | | | Q¦(y₂)                                                    ASSUME
49  | | | | | | | | | | *show Q¦(z₂)
50  | | | | | | | | | |  ¬Q¦(z₂)                                                  ASSUME
51  | | | | | | | | | |  *show (Q¦(z₂)→Q¦(z₂))
52  | | | | | | | | | |  | Q¦(z₂)                                                 ASSUME
53  | | | | | | | | | |  | *show Q¦(z₂)
54  | | | | | | | | | |  | | ¬Q¦(z₂)                                              ASSUME
55  | | | | | | | | | |  | | *show (P¦(z₂)↔P¦(z₂))
56  | | | | | | | | | |  | | | *show (P¦(z₂)→P¦(z₂))
57  | | | | | | | | | |  | | | | P¦(z₂)                                           ASSUME
58  | | | | | | | | | |  | | | | *show P¦(z₂)
59  | | | | | | | | | |  | | | | | ¬P¦(z₂)                                        ASSUME
60  | | | | | | | | | |  | | | | | *show (P¦(y₂)↔P¦(z₂))
61  | | | | | | | | | |  | | | | | | *show (P¦(z₂)→P¦(y₂))
62  | | | | | | | | | |  | | | | | | | P¦(z₂)                                   ASSUME
63  | | | | | | | | | |  | | | | | | | *show (P¦(z₂)↔P¦(z₂))
64  | | | | | | | | | |  | | | | | | | | *show (P¦(z₂)→P¦(z₂))
65  | | | | | | | | | |  | | | | | | | | | P¦(z₂)                                 ASSUME
66  | | | | | | | | | |  | | | | | | | | | *show P¦(z₂)
67  | | | | | | | | | |  | | | | | | | | | | ¬P¦(z₂)                              ASSUME
68  | | | | | | | | | |  | | | | | | | | | | *show (Ex₂)P¦(x₂)
69  | | | | | | | | | |  | | | | | | | | | | | (Ex₂)P¦(x₂)                        65,EG
70  | | | | | | | | | |  | | | | | | | | | | (Ay₂)Q¦(y₂)                          9,68,MP
71  | | | | | | | | | |  | | | | | | | | | | P¦(z₂)                               68,EI
72  | | | | | | | | | |  | | | | | | | | | | ¬(Ay₂)(Q¦(z₂)→Q¦(y₂))                11,UI
73  | | | | | | | | | |  | | | | | | | | | | ¬(Ay₂)(P¦(z₂)↔P¦(y₂))                22,UI
74  | | | | | | | | | |  | | | | | | | | | | Q¦(z₂)                               70,UI
75  | | | | | | | | | |  | | | | | | | | | | (Ey₂)¬(Q¦(z₂)→Q¦(y₂))                72,QN
76  | | | | | | | | | |  | | | | | | | | | | (Ey₂)¬(P¦(z₂)↔P¦(y₂))                73,QN
77  | | | | | | | | | |  | | | | | | | | | | ¬(Q¦(z₂)→Q¦(z₂))                     75,EI
78  | | | | | | | | | |  | | | | | | | | | | ¬(P¦(z₂)↔P¦(y₂))                     76,EI
79  | | | | | | | | | |  | | | | | | | | | | *show (Q¦(z₂)→Q¦(z₂))
80  | | | | | | | | | |  | | | | | | | | | | | Q¦(z₂)                             70,UI
81  | | | | | | | | | |  | | | | | | | | | *show (P¦(z₂)→P¦(z₂))
82  | | | | | | | | | |  | | | | | | | | |  P¦(z₂)                                 ASSUME
83  | | | | | | | | | |  | | | | | | | | |  *show P¦(z₂)
84  | | | | | | | | | |  | | | | | | | | |  | ¬P¦(z₂)                              ASSUME
85  | | | | | | | | | |  | | | | | | | | |  | *show (Ex₂)P¦(x₂)
86  | | | | | | | | | |  | | | | | | | | |  | | (Ex₂)P¦(x₂)                      82,EG
87  | | | | | | | | | |  | | | | | | | | |  | (Ay₂)Q¦(y₂)                          9,85,MP
88  | | | | | | | | | |  | | | | | | | | |  | P¦(y₂)                               85,EI
89  | | | | | | | | | |  | | | | | | | | |  | ¬(Ay₂)(Q¦(y₂)→Q¦(y₂))                11,UI
90  | | | | | | | | | |  | | | | | | | | |  | ¬(Ay₂)(P¦(y₂)↔P¦(y₂))                22,UI
91  | | | | | | | | | |  | | | | | | | | |  | Q¦(y₂)                               87,UI
92  | | | | | | | | | |  | | | | | | | | |  | (Ey₂)¬(Q¦(y₂)→Q¦(y₂))                89,QN
93  | | | | | | | | | |  | | | | | | | | |  | (Ey₂)¬(P¦(y₂)↔P¦(y₂))                90,QN
94  | | | | | | | | | |  | | | | | | | | |  | ¬(Q¦(y₂)→Q¦(y₂))                     92,EI
```

```
95  | | |
```

```
142 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | (Ey1)¬(P¦(xa)↔P¦(y1))                    139,QN
143 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ¬(Q¦(xa)→Q¦(x1))                         141,EI
144 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ¬(P¦(xa)↔P¦(xa))                         142,EI
145 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | *show (Q¦(xa)→Q¦(x1))
146 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Q¦(x1)                                   136,UI
147 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | (P¦(za)↔P¦(z1))                          130,113,CB
148 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ¬(P¦(za)↔P¦(z1))                         46,R
149 | | | | | | | | | | | | | | | | | | | | | | | | | | | | *show (P¦(y1)→P¦(z1))
150 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | P¦(y1)                                   ASSUME
151 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ¬P¦(y1)                                  33,R
152 | | | | | | | | | | | | | | | | | | | | | | | | | | | | (P¦(y1)↔P¦(z1))                          149,110,CB
153 | | | | | | | | | | | | | | | | | | | | | | | | | | | | ¬(P¦(y1)↔P¦(z1))                         45,R
154 | | | | | | | | | | | | | | | | | | | | | | | | | | | (P¦(za)↔P¦(z1))                          105,56,CB
155 | | | | | | | | | | | | | | | | | | | | | | | | | | | ¬(P¦(za)↔P¦(z1))                         25,R
156 | | | | | | | | | | | | | | | | | | | | | | | | | | ¬(Q¦(za)→Q¦(za))                         44,R
157 | | | | | | | | | | | | | | | | | | | | | | | | | | ((Ex2)Q¦(x2)↔(Ay2)P¦(y2))                 29,27,CB
158 | | | | | | | | | | | | | | | | | | | | | | | | | ((Ex2)Q¦(x2)↔(Ay2)P¦(y2))                 7,19,MP
159 | | | | | | | | | | | | | | | | | | | | | | | | | (Ay1)(P¦(xa)↔P¦(y1))                      19,EI
160 | | | | | | | | | | | | | | | | | | | | | | | | | ¬(Ay1)(Q¦(xa)→Q¦(y1))                     11,UI
161 | | | | | | | | | | | | | | | | | | | | | | | | | (P¦(xa)↔P¦(za))                           159,UI
162 | | | | | | | | | | | | | | | | | | | | | | | | | (P¦(xa)↔P¦(xa))                           159,UI
163 | | | | | | | | | | | | | | | | | | | | | | | | | ((Ex2)Q¦(x2)→(Ay1)P¦(y1))                 158,BC
164 | | | | | | | | | | | | | | | | | | | | | | | | | ((Ay2)P¦(y2)→(Ex1)Q¦(x1))                 158,BC
165 | | | | | | | | | | | | | | | | | | | | | | | | | (P¦(xa)→P¦(za))                           161,BC
166 | | | | | | | | | | | | | | | | | | | | | | | | | (P¦(za)→P¦(xa))                           161,BC
167 | | | | | | | | | | | | | | | | | | | | | | | | | (P¦(xa)→P¦(xa))                           162,BC
168 | | | | | | | | | | | | | | | | | | | | | | | | | (Ey1)¬(Q¦(xa)→Q¦(y1))                     160,QN
169 | | | | | | | | | | | | | | | | | | | | | | | | | ¬(Q¦(xa)→Q¦(xa))                          168,EI
170 | | | | | | | | | | | | | | | | | | | | | | | | | *show (Q¦(xa)→Q¦(xa))
171 | | | | | | | | | | | | | | | | | | | | | | | | | | Q¦(xa)                                    ASSUME
172 | | | | | | | | | | | | | | | | | | | | | | | | | | *show Q¦(xa)
173 | | | | | | | | | | | | | | | | | | | | | | | | | | | ¬Q¦(xa)                                   ASSUME
174 | | | | | | | | | | | | | | | | | | | | | | | | | | | *show (Ex2)P¦(x2)
175 | | | | | | | | | | | | | | | | | | | | | | | | | | | | ¬(Ex2)P¦(x2)                              ASSUME
176 | | | | | | | | | | | | | | | | | | | | | | | | | | | | ¬(Ay2)Q¦(y2)                              175,10,MT
177 | | | | | | | | | | | | | | | | | | | | | | | | | | | | (Ax2)¬P¦(x2)                              175,QN
178 | | | | | | | | | | | | | | | | | | | | | | | | | | | | (Ey2)¬Q¦(y2)                              176,QN
179 | | | | | | | | | | | | | | | | | | | | | | | | | | | | ¬P¦(zb)                                   177,UI
180 | | | | | | | | | | | | | | | | | | | | | | | | | | | | ¬P¦(xb)                                   177,UI
181 | | | | | | | | | | | | | | | | | | | | | | | | | | | | ¬Q¦(wb)                                   178,EI
182 | | | | | | | | | | | | | | | | | | | | | | | | | | | | ¬(Ay1)(Q¦(wb)→Q¦(y1))                     11,UI
183 | | | | | | | | | | | | | | | | | | | | | | | | | | | | (P¦(xb)↔P¦(wb))                           159,UI
184 | | | | | | | | | | | | | | | | | | | | | | | | | | | | ¬P¦(wb)                                   177 UI
185 | | | | | | | | | | | | | | | | | | | | | | | | | | | | (P¦(xb)→P¦(wb))                           183,BC
186 | | | | | | | | | | | | | | | | | | | | | | | | | | | | (P¦(wb)→P¦(xb))                           183,BC
187 | | | | | | | | | | | | | | | | | | | | | | | | | | | | (Ey1)¬(Q¦(wb)→Q¦(y1))                     182,QN
188 | | | | | | | | | | | | | | | | | | | | | | | | | | | | ¬(Q¦(wb)→Q¦(wb))                          187,EI
```

```
189 | | | | | | | | *show (Q¦(w₀)→Q¦(w₀))                    ASSUME
190 | | | | | | | |  Q¦(w₀)                                  181,R
191 | | | | | | | |  ¬Q¦(w₀)                                 9,174,MP
192 | | | | | | (Ay₀)Q¦(y₀)                                  192,UI
193 | | | | | | Q¦(x₀)
```

CPU time: 1102 msec
Statements: 80012

X. APPENDIX II: A FLOW CHART OF THINKER's PROOF STRATEGY

This appendix gives an informal flow-chart explanation of the proof strategies used by THINKER. We start with a discursive explanation of ONESTEP and SIMPLEPROOF, follow this with the description of initialization, and then go on to the description of PROOF. Certain footnotes to various steps in PROOF give a more discursive explanation of the steps.

A. ONESTEP and SIMPLEPROOF

ONESTEP(ø) -- use formula ø to prove the most recent goal [globally given] in one step. This looks to the goal and ø, and decides what kind of formula must be in [the global] ANTELINES to yield a proof of the goal using ø. If it finds such an antecedent line, the function introduces the pair <goal,annotation> into ANTELINES and returns **true**. (ONESTEP might introduce complexity. If the goal is, for example, (P+Q) and ø=P, then ONESTEP(P) introduces <(P+Q), line# ADD> into ANTELINES, where the formula (P+Q) is more complex than the one from which it was generated, P. But the complexity is only "one level higher" than ø, and in any case ONESTEP always terminates the proof at a given show level.)

SIMPLEPROOF(ø) -- find a very simple proof of ø from the [global] ANTELINES. If found, it introduces the pair <ø,annotation> into ANTELINES and returns **true**. (SIMPLEPROOF might introduce complexity. If ø=(P+Q) and

P is in ANTELINES, then SIMPLEPROOF will introduce <(P+Q), line# ADD> into ANTELINES). Unlike ONESTEP, SIMPLEPROOF might use templates. If ø=P, SIMPLEPROOF will look for (@→P), among others, and if found see whether @'s token is also in ANTELINES.

Note that whenever a formula is added to ANTELINES, whether by these functions or others, all the templates are also added, and the variable and constant tables are updated. The reverse (deletion) is done whenever a line is removed from ANTELINES. The relevant functions are ADDANTE(ø) and DELANTE(ø).

B. Initialization

(I here consider the case of theorems without premises. Extra things happen to premises which would only obscure the essential points being outlined here.)

```
PRMAT := NIL {proof table}
GOALST := NIL {goal stack}
ANTELINES := NIL {antecedent lines table}
CURLINE := 1 {global current line in PRMAT}
IND := false {flag for indirect proof}
TRY := false {flag for CHAINING strategies}
CURLEVEL := 0 {global current depth of goal stack}
```

Initial call takes the form:**

```
READ(TO.PROVE);
N := CURLINE;
PROOF(TO.PROVE,N);
```

**Note that N (the line that this subproof starts on) is called by value. At the end of an embedded subproof, this line number will be available for boxing and cancelling. See step 11.

C. PROOF

The description of PROOF(ø,n) follows. Formula ø is to be proved, and its line number in PRMAT is n.

A. CURLINE := CURLINE+1; CURLEVEL := CURLEVEL+1;

B. PUSH(ø,GOALST) {add ø to the goalstack}
PRMAT[n,1] := 'show' ø

C. if SIMPLEPROOF(ø) then GOTO 11
{SIMPLEPROOF has added ø to proof matrix, increments CURLINE; step 11 wraps up proof}

D. (splitting heuristics)

a. if ø=(θ↔ϕ) then

i) if (ϕ→θ) ∉ GOALST then

a) PROOF((ϕ→θ),CURLINE)

b) if ONESTEP((ϕ→θ)), GOTO 11

ii) if (θ→ϕ) ∉ GOALST then PROOF((θ→ϕ))

iii)

if SIMPLEPROOF((θ↔ϕ)) then GOTO 11''

b. if ø=(θ&ϕ) then

i) if θ ∉ GOALST then

a) PROOF(θ,CURLINE)

b) if ONESTEP(θ) then GOTO 11

ii) if ϕ ∉ GOALST then PROOF(ϕ,CURLINE)

iii)

if SIMPLEPROOF((θ&ϕ)) then GOTO 11

'' SIMPLEPROOF is guaranteed to succeed here (and in other places like this when it is called) because the previous two steps have put the corresponding conditionals into ANTELINES, so that SIMPLEPROOF will find them and add the ↔ formula. This makes it possible to GOTO 11 and wrap up the proof.

c. if $\phi=(A\alpha)\Vdash$ and $\neg$FREE(α) and $\Vdash \notin$ GOALST then[50]

i) PROOF($\Vdash$,CURLINE)

ii) GOTO 11

---

[50] FREE(α) checks whether α is free in antecedent lines. Recall that in Kalish & Montague, "universal derivation" takes the place of universal generalization. I.e., if α is not free in antecedent lines then

show $(A\alpha)\Vdash$

X_1

.

.

X

can be boxed of $\Vdash$ occurs unboxed amongst $X_1 \ldots X$ and there are no uncancelled Show's amongst them.

```
E.  (Assumptions: recall that ø is the line being proved)
    a.  if ø=¬ψ and ψ ∉ ANTELINES then
        i)   PRMAT(CURLINE,1) := ψ;
        ii)  PRMAT(CURLINE,2) := "ASSUME";
        iii)
             CURLINE := CURLINE+1;
        iv)  ADDANTE(ψ);
        v)   if ONESTEP(ψ) then GOTO 11;
        vi)  IND := true;
    b.  if ø=(ψ→θ) and ψ ∉ ANTELINES then
        i)   if SIMPLEPROOF(θ) then GOTO 11;
        ii)  PRMAT(CURLINE,1) := ψ;
        iii)
             PRMAT(CURLINE,2) := "ASSUME";
        iv)  CURLINE := CURLINE+1;
        v)   ADDANTE(ψ);
        vi)  if ONESTEP(ψ) then GOTO 11;
        vii)
             PROOF(θ,CURLINE);
        viii)
             GOTO 11;
    c.  if ¬ø ∉ ANTELINES then
        i)   PRMAT(CURLINE,1) := ¬ø;
        ii)  PRMAT(CURLINE,2) := "ASSUME";
        iii)
             CURLINE := CURLINE+1;
        iv)  ADDANTE(¬ø);
```

v) if ONESTEP(¬ø) then GOTO 11;

vi) IND := true;

F. (Forward inference)[51]

a. OLDCUR := CURLINE;

b. if FINDQN then [if ONESTEP(PRMAT(CURLINE,1))[52] then GOTO 11 else GOTO 6b {more QNs}]

c. if FINDDN then [if ONESTEP(PRMAT(CURLINE,1)) then GOTO 11 else GOTO 6c {more DNs}]

d. if FINDBC then [if ONESTEP(PRMAT(CURLINE,1)) then GOTO 11 else GOTO 6d {more BCs}]

e. if FINDS then [if ONESTEP(PRMAT(CURLINE,1)) then GOTO 11 else GOTO 6e {more SIMPs}]

f. if FINDMP then [if ONESTEP(PRMAT(CURLINE,1)) then GOTO 11 else GOTO 6f {more MPs}]

g. if FINDMT then [if ONESTEP(PRMAT(CURLINE,1)) then GOTO 11 else GOTO 6g {more MTs}]

h. if FINDMTP then [if ONESTEP(PRMAT(CURLINE,1)) then GOTO 11 else GOTO 6h {more MTPs}]

i. if OLDCUR≠CURLINE then GOTO 6a[53]

j. FINDALLEI {Existentially instantiate all lines you can. Mark these lines as being EIed under this CURLEVEL. Do not EI again under this level.}[54]

---

[51] FINDQN pushes negations to the *inside* of quantifiers. It does not implement the reverse type of QN. FINDDN *deletes* double negation; it does not add them. Thus all the propositional and quantifier negation rules "simplify" (not add complexity) and so will eventually terminate.

[52] This will be a ONESTEP on the line just added to the proof matrix by ONESTEP.

[53] Each time one of the FINDs discovers an inference, the line is added to PRMAT with an appropriate annotation and CURLINE is incremented. This step i is a means to go back to try the FIND rules on the results of previous FINDs.

[54] FINDALLEI will only work once on a given formula under a given CURLEVEL. So it terminates.

k. if FINDUI then [if ONESTEP(PRMAT(CURLINE,1)) then GOTO 11 else GOTO 6k {more UIs}][55]

l. if OLDCUR ≠ CURLINE then GOTO 6a[56]

---

[55]FINDUI is instantiated in the following way: (a) FINDUI maintains and updates A-lists. (b) An EI might put a variable on the P-list. If the existentially quantified line has a universally quantified line on its A-list, then that universally quantified line is marked "bad". No "bad" line can be UIed to any variable on the P-list by FINDUI.

[56]So, in general, the FORWARD INFERENCE applies all the rules of inference that "simplify" formulae. This section is finite, therefore, and will eventually terminate. So the two ways to exit FORWARD INFERENCE are: (a) Some ONESTEP succeeds, (b) No more rules can be applied to any ANTELINES.

G. (TRYNEG) if IND then

a. if $\neg(\theta\rightarrow\psi) \in$ ANTELINES and $(\theta\rightarrow\psi) \notin$ GOALST then

i) PROOF($(\theta\rightarrow\psi)$,CURLINE)

ii) if ONESTEP($(\theta\rightarrow\psi)$) then GOTO 11 ''

b. if $\neg(\theta\leftrightarrow\psi) \in$ ANTELINES and $(\theta\leftrightarrow\psi) \notin$ GOALST then

i) PROOF($(\theta\leftrightarrow\psi)$,CURLINE)

ii) if ONESTEP($(\theta\leftrightarrow\psi)$) then GOTO 11

c. if $\neg(\theta\&\psi) \in$ ANTELINES and $(\theta\&\psi) \notin$ GOALST then

i) PROOF($(\theta\&\psi)$,CURLINE)

ii) if ONESTEP($(\theta\&\psi)$) then GOTO 11

d. if $\neg(\theta+\psi) \in$ ANTELINES then

i) if $\theta \notin$ GOALST then

a) PROOF(θ,CURLINE);

b) PRMAT(CURLINE,1) := $(\theta+\psi)$;

c) PRMAT(CURLINE,2) := "Add"

d) CURLINE := CURLINE+1;

e) if ONESTEP($(\theta+\psi)$) then go to 11

ii) else if $\psi \notin$ GOALST then

a) PROOF(ψ,CURLINE);

b) PRMAT(CURLINE,1) := $(\theta+\psi)$;

c) PRMAT(CURLINE,2) := "Add"

d) CURLINE := CURLINE+1;

e) if ONESTEP($(\theta+\psi)$) then go to 11

'' ONESTEP in this and the following must succeed, since there must be an explicit contradiction here. I put it this way because in the Kalish & Montague system, to cancel by a contradiction, both halves of the contraditions must be "below" the line cancelled. If this is not otherwise so, ONESTEP will Repeat the appropriate line.

H. (Chaining)

a. if $(\theta \rightarrow \phi) \in$ ANTELINES and $\theta \notin$ GOALST and $\phi \notin$ ANTELINES then

i) PROOF(θ,CURLINE);

ii) GOTO 6;

b. if $(\theta \rightarrow \phi) \in$ ANTELINES then

i) if $\phi \notin$ ANTELINES and $\neg\theta \notin$ GOALST then

a) PROOF($\neg\theta$,CURLINE);

b) GOTO 6;

ii) if $\theta \notin$ ANTELINES and $\neg\phi \notin$ GOALST then

a) PROOF($\neg\phi$,CURLINE);

b) GOTO 6;

I. (UIPROHIB)

a. OLDCUR := CURLINE;

b. UIPROHIB();

c. if OLDCUR≠CURLINE then GOTO 6 else GOTO 10

J. (HELP)

a. Print(PRMAT);

b. Read(X);

c. if X = "Show" ϕ then

i) PROOF(ϕ,CURLINE);

ii) GOTO 6;

else

i) PRMAT(CURLINE,1) := ϕ;

ii) PRMAT(CURLINE,2) := "GOD says so";

iii)

CURLINE := CURLINE+1;

```
        iv) GOTO 6;
K.  (Wrap up proof)
    a.  for i:=n+1 until CURLINE do
        i)  DELANTE(PRMAT(i,1));
        ii) Prefix '|' to PRMAT(i,1);
    b.  ADDANTE(PRMAT(n,1));
    c.  Prefix '*' to PRMAT(n,1);
    d.  CURLEVEL := CURLEVEL-1;
```

XI. REFERENCES

Anderson, R. "Completeness of the Locking Restriction for Paramodulation" Dept. Comp. Sci., Univ. Texas at Austin.

Appel, K. & Hakin, W. (1977) "The Solution of the Four-Color Map Problem" *Scientific American* 137: 108-121.

Ballantyne, M. (1975) "Computer Generation of Counterexamples in Topology" Univ. Texas Tech. Document.

Bibel, W. (1979) "Syntax Directed, Semantic Supported Program Synthesis" *4th Conference on Automated Deduction* (Austin: Univ. Texas Press) pp. 140-147.

Bibel, W. (1982) "A Comparative Study of Several Proof Procedures" *Artificial Intelligence* 18: 269-294.

Bierman, A. (1976) "Approaches to Automatic Programming" in *Advances in Computers*, Vol. 15 (N.Y: Academic Press)

Bledsoe, W. (1971) "Splitting and Reduction Heuristics in Automatic Theorem Proving" *Artificial Intelligence* 2: 55-78.

Bledsoe, W. (1977) "Non-Resolution Theorem Proving" *Artificial Intelligence* 9: 1-36.

Bledsoe, W., Boyer, R. & Henneman, W. (1972) "Computer Proofs of Limit Theorems" *Artificial Intelligence* 3: 27-60.

Bledsoe, W. & Bruell, P. (1974) "A Man-Machine Theorem Proving System" *Artificial Intelligence*, 5: 51-72.

Bobrow, D. & Raphael, B. (1974) "New Programming Languages

for Artificial Intelligence Research" *ACM Computing Surveys* 6: 153-174.

Boyer, R. (1971) *Locking: A Restriction of Resolution*. Ph.D. Thesis, Univ. Texas, Austin.

Boyer, R. & Moore, J. (1979) *A Computational Logic* (N.Y.: Academic Press).

Brown, F. (1979) "A Theorem Prover for Meta Theory" *Proc. of the Fourth Workshop on Automated Deduction*. (Austin: Univ. Texas Press).

Bundy, Alan (1977) "Can Domain Specific Knowledge be Generalized?" *IJCAI* 5: 496-502.

Chang, C-L. (1970) "The Unit Proof and the Input Proof in Theorem Proving" *JACM* 17: 698-707.

Chang, C-L. (1977) "Resolution Plans in Theorem Proving" *IJCAI* 5: 143-148.

Chang, C-L & Lee, R. (1973) *Symbolic Logic and Mechanical Theorem Proving* (N.Y.: Academic Press).

Church, A. (1956) *Introduction to Mathematical Logic* (Princeton: Princeton UP).

Davis, M. & Putnam, H. (1960) "A Computing Procedure for Quantification Theory" *JACM* 7: 201-215.

de Champeaux, D. (1979) "Sub-Problem Finder and Instance Checker: Two Cooperating Preprocessors for Theorem Provers" *IJCAI* 6: 191-196.

Derksen, J., Rulifson, J., & Waldinger, R. (1972) "The QA4 Language Applied to Robot Planning" *Proc. Fall Joint Computer Conference* Vol. 41 #2, 1181-1192.

Elschlager, R. & Phillips, J. (1979) "Automatic Programming" Computer Science Report CS-79-758, Stanford. To appear in A. Barr & E. Feigenbaum *Handbook of Artificial Intelligence* Vol. III.

Ernst, G.W. (1971) "The Utility of Independent Subgoals in Theorem Proving" *Information and Control* 18: 237-252.

Ernst, G. & Newell, A. (1969) *GPS: A Case Study of Generality and Problem Solving* (N.Y.: Academic Press).

Feigenbaum, E. & Feldman, J. (1963) *Computers and Thought* (N.Y.: McGraw-Hill).

Fikes, R. & Nilsson (1971) "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving" *Artificial Intelligence* 2: 189-208.

Gazdar, G. (1981) "Unbounded Dependencies and Coordinate Structure" *Linguistic Inquiry* 12: 155-184.

Gazdar, G. Klein, E., Pullum, G., & Sag, I. (forthcoming) *English Syntax*.

Gelernter, H. (1963) "Realization of a Geometry Theorem Proving Machine" In Feigenbaum, E. & Feldman, J. *Computers and Thought* (N.Y.: McGraw Hill) pp. 134-152.

Georgacarakos, G. & Smith, R. (1978) *Elementary Formal Logic* (N.Y.: McGraw Hill).

Goad, C. (1980) "Proofs as Descriptions of Computation" *5th Conference on Automatic Deduction* (Berlin: Springer-Verlag) pp. 39-52.

Goguen, J.A. (1980) *How to Prove Inductive Hypotheses without Induction*. Technical Report, SRI International.

Green, C. (1969) "Theorem Proving by Resolution as a Basis for Question Answer Systems" in B. Meltzer & D. Michie (ed.) *Machine Intelligence 4* (Edinburgh: Edinburgh Univ. Press) pp. 183-205.

Griswold, R., Poage, J., & Polonsky, I. (1968) *The Snobol4 Programming Language*. (Englewood Cliffs: Prentice-Hall).

Grice, H.P. (1975) "Logic and Conversation" in P. Cole & J. Morgan (eds.) *Syntax and Semantics, III: Speech Acts* (N.Y.: Academic Press) pp. 41-58.

Guard, J., Oglesby, F., Bennett, J. & Settle, L. (1969) "Semi Automated Mathematics" *JACM* 16: 49-62.

Guiho, G. & Gresse, C. (1980) "Program Synthesis from Incomplete Specification" in *5th Conference on Automatic Deduction* (Berlin: Springer-Verlag) pp. 53-62.

Harrison, M. (1977) "A Hierarchical Approach to Theorem Proving" *IJCAI* 5: 529-535.

Henschen, L. (1975) "Semantic Resolution of Horn Sets" *IJCAI 4* pp. 46-52.

Henschen, L. & Evangelist, W. (1977) "Theorem Proving by Covering Expressions" *IJCAI* 5: 541-542.

Henschen, L. & Wos, L. (1974) "Unit Refutations and Horn Sets" *JACM* 21: 590-605.

Jeffrey, R. (1967) *Formal Logic* (Englewood Cliffs: Prentice-Hall).

Kalish, D. & Montague, R. (1964) *Logic* (N.Y.: World, Hartcourt & Brace).

Kowalski, R. (1974) "A Proof Procedure Using Connection

Graphs" *JACM* 22: 572-595.

Kowalski, R. (1978) *Logic For Problem Solving*. (New York: North-Holland).

Lehnert, W. (1977) *The Process of Question Answering* (Yale Univ., Computer Science Report #88).

Loveland, D. (1968) "Mechanical Theorem Proving by Model Elimination" *JACM* 15: 236-251.

Loveland, D. (1970) "A Linear Format for Resolution" *Proc. IRIA Symp. Automatic Demonstration, Versailles 1968.* (N.Y.: Springer-Verlag).

Loveland, D. (1978) *Automated Theorem Proving* (Amsterdam: North-Holland).

Luckham, D. (1970) "Refinements in Resolution Theory" *Proc. IRIA Symp. Automatic Demonstration, Versailles 1968.* (N.Y.: Springer-Verlag).

Manna, Z. & Waldinger, R. (1977) *Studies in Automatic Programming Logic* (N.Y.: Academic Press).

Martelli, A. (1977) "Theorem Proving with Structure Sharing and Efficient Unification" *IJCAI* 5: 543.

McCawley, J. (1981) *Everything That Linguists Have Always Wanted to Know About Logic.* (Chicago: Univ. Chicago Press).

Meltzer, B. (1966) "Theorem Proving for Computers: Some Results on Resolution and Renaming" *Computing Journal* 8: 341-343.

Montague, R. (1970) "English as a Formal Language" in Thomason (1974) pp. 188-221.

Montague, R. (1973) "The Proper Treatment of Quantification in English" in Thomason (1974) pp. 247-270.

Murray, N. (1982) "Completely Non-Clausal Theorem Proving" *Artificial Intelligence* 18: 67-86.

Nevins, A. (1974) "A Human Oriented Logic for Automatic Theorem Proving" *JACM* 21: 606-621.

Newell, A., Shaw, J., & Simon, H. (1957) "Empirical Explorations with the Logic Theory Machine: A Case Study in Heuristics" in Feigenbaum, E. & Feldman, J. *Computers and Thought* (N.Y.: McGraw-Hill) pp. 109-133.

Newell, A. & Simon, H. (1960) "GPS: A Program that Simulates Human Thought" in E. Feigenbaum & J. Feldman (eds.) *Computers and Thought* (N.Y.: McGraw-Hill) 1963, pp. 279-293.

Newell, A. & Simon, H. (1972) *Human Problem Solving* (Englewood Cliffs: Prentice-Hall).

Nilsson, N. (1980) *Principles of Artificial Intelligence* (Palo Alto: Tioga Pub. Co).

Norton, L. (1971) "Experiments with a Heuristic Theorem-Proving Program for Predicate Calculus with Identity" *Artificial Intelligence* 2: 261-284.

Pelletier, F.J. & Wilson, D. (1982) "Heuristic Theorem Proving" in W. Maxwell *Thinking: An Interdisciplinary Approach*. (Philadelphia: Franklin Press) forthcoming.

Reiter, R. (1973) "A Semantically Guided Deductive System for Automatic Theorem Proving" *IJCAI 3* pp. 41-46.

Reiter, R. (1978) "On Reasoning by Default" *TINLAP* 2:

210-218.

Robinson, J. (1965) "A Machine Oriented Logic Based on the Resolution Principle" *JACM* 12: 23-41.

Robinson, J. (1968) "The Generalized Resolution Principle" in Dale & Michie (eds) *Machine Intelligence 3* (Edinburgh: Oliver and Boyd) pp. 77-93.

Sacerdoti, E. (1974) "Planning in a Hierarchy of Abstraction Spaces" *Artificial Intelligence* 5: 115-136.

Sag, I. (1981) "A Semantic Theory of 'NP Movement'" in Jacobson, P. & Pullum, G. (eds) *The Nature of Syntactic Representation* (Dortrecht: Reidel) pp. ??

Sandford, D. (1980) *Using Sophisticated Methods in Resolution Theorem Proving. Lecture Notes in Computer Science* 90, (N.Y.: Springer-Verlag).

Schubert, L. & Pelletier, F. (1982) "From English to Logic: Context-Free Computation of 'Conventional' Logic Translations" *Jour. ACL* (forthcoming).

Siklóssy, L., A. Rich, & V. Marinov (1973) "Breadth First Search: Some Surprising Results" *Artificial Intelligence* 4: 1-27.

Slagle, J. (1965) "A Proposed Preference Strategy Using Sufficiency Resolution for Answering Questions" UCRL-14361, Lawrence Radiation Lab: Livermore, Cal.

Slagle, J. (1967) "Automatic Theorem Proving with Renamable and Semantic Resolution" *JACM* 14: 687-697.

Slagle, J. & Farrell, C. (1971) "Experiments in Automatic Learning for a Multipurpose Heuristic Program" *Comm. ACM*

14: 91-99.

Slagle, J. & Norton, L. (1971) "Experiments with an Automated Theorem Prover Having Partial Ordering Rules" Div. of Comp. Res. and Tech., Nat. Inst. of Health, Bethesda, Maryland.

Stefferud, E. (1963) "The Logic Theory Machine: A Model Heuristic Program" RAND Memo RM-3731-CC, Santa Monica.

Thomason, R. (1970) *Symbolic Logic: An Introduction.* (Toronto: Macmillan).

Thomason, R. (1974) *Formal Philosophy*. (New Haven: Yale UP).

Wang, H. (1963) "Toward Mechanical Mathematics" in Sayre, K. & Crosson, P. *The Modelling of Mind* (Notre Dame: Notre Dame UP) pp. 91-120.

Whitehead, A. & Russell, B. (1910-1912) *Principia Mathematica, Vols. I-III* (Cambridge: Cambridge UP).

Wos, L., Robinson, G., and Carson, D. (1965) "Efficiency and Completeness of the Set of Support Strategy in Theorem Proving" *JACM* 12: 536-541.